UiO : Department of Informatics
University of Oslo

Robust estimation of churn in privacy-preserving P2P networks

David Kai Christen Kristensen
Master's Thesis Autumn 2013

Robust estimation of churn in privacy-preserving P2P networks

David Kai Christen Kristensen

15th November 2013

Abstract

A number of recently proposed techniques for aggregating data in P2P networks suffer in performance under the presence of network churn and the recently proposed solutions for estimating churn are unable to do so in a privacy-preserving manner. Following this observation, we present a method for a robust aggregation method which allows for estimating churn in an efficient and accurate manner, without compromising node privacy. Essentially we are taking gossiping and selection techniques usually used to create network overlays, and use them to aggregate data which can be used for estimation. The experimental evaluation shows that our approach is able to estimate a reasonable correct churn in a wide range of churn scenarios without compromising node privacy.

Keywords: Aggregation, estimation, churn, peer-to-peer, friend-to-friend, privacy-preservation, robust, dissemination.

Contents

I Introduction, background and application context **1**

1 Introduction **3**

2 Background **7**
2.1 Peer to peer networks 7
2.1.1 Structured P2P networks 8
2.1.2 Unstructured P2P networks 8
2.2 Friend to friend networks 9
2.3 Gossiping 9
2.4 Churn and models for churn 9
2.5 Random graph 10
2.6 Standard deviation 11
2.7 Cumulative distribution function 11

3 Related work **13**
3.1 Distributed aggregation 13
3.1.1 Gossip-based aggregation 13
3.2 Churn estimation 15
3.2.1 Estimating leaves and joins 15
3.2.2 Estimating T_{on} and T_{off} 16
3.3 Evaluating privacy in networks 17

4 Application context **19**
4.1 A robust privacy-preserving protocol for data dissemination 19
4.2 Privacy guarantees 20
4.3 Overview of the protocol 21
4.3.1 Creating and removing pseudonyms 22
4.3.2 Gossiping pseudonyms 22
4.3.3 Selecting pseudonym links 23

II Problem statement, solution and evaluation **25**

5 Problem statement **27**
5.1 Pseudonym lifetime and privacy 27
5.2 Estimating mean T_{off} time in a privacy-preserving manner . 27
5.2.1 Formal problem statement 28

6 Solution **29**
6.1 Design considerations . . . 29
6.2 Assumptions . . . 30
6.3 Solution . . . 31
6.3.1 Initial state of a node . . . 32
6.3.2 Updating O . . . 32
6.3.3 Calculating E . . . 32
6.3.4 Dealing with expired pseudonyms . . . 32
6.3.5 Gossiping . . . 32
6.4 Reasoning behind design choices . . . 34
6.4.1 Choosing a selection . . . 34
6.4.2 How long to keep values from offline nodes . . . 34
6.4.3 Disseminating and updating values . . . 35
6.4.4 Estimation . . . 35
6.5 Added computational and message complexity . . . 36
6.6 Usage outside the application context . . . 37
6.7 Summary . . . 37

7 Evaluation **39**
7.1 Experimental environment . . . 39
7.1.1 Trust graph . . . 39
7.1.2 Churn . . . 40
7.1.3 Default configuration parameters for proposed solution 41
7.1.4 Performance metrics for estimation . . . 42
7.2 Experiments . . . 43
7.2.1 Connectivity . . . 44
7.2.2 Mean path length . . . 45
7.2.3 Performance under artificial churn while self monitoring . . . 45
7.2.4 Pseudonym lifetimes effect on accuracy . . . 49
7.2.5 Overlay sizes effect on accuracy . . . 50
7.2.6 Performance under real-life churn using Skype traces 52
7.3 Summary of estimation performance . . . 54
7.4 Privacy-preservation . . . 54
7.4.1 Performance metrics for privacy-preservation . . . 54
7.5 Evaluating privacy-preservation . . . 55
7.5.1 Set strength . . . 55
7.5.2 Privacy-preservation under artificial churn . . . 56
7.5.3 Privacy-preservation under real-life churn using Skype traces . . . 57
7.6 Evaluation summary . . . 58

III Conclusion and future work **59**

8 Conclusion and future work **61**
8.1 Conclusion . . . 61
8.2 Future work . . . 62

8.3 Acknowledgements 63

List of Figures

2.1 CDF example: CDF distribution of availability 11

4.1 Example of a trust graph and derived communication overlay presented in [33] 20
4.2 Architecture for privacy-preserving data dissemination presented in [33] 21

7.1 CDF distribution of availability for nodes used in experiments 41
7.2 CDF distribution of node availability in Skype traces from [31] 41
7.3 Node connectivity over time 44
7.4 Mean connectivity for the duration of the experiments 45
7.5 Normalized mean path length for the duration of the experiments 45
7.6 Convergence of proposed algorithm while self-monitoring . 46
7.7 Distribution of estimate over own value for all nodes 47
7.8 Distribution of estimated means over own value for online nodes 48
7.9 Distribution of estimated means over own value for all nodes when including own estimate in the average 49
7.10 Convergence of proposed algorithm while self-monitoring, with different pseudonym lifetime ratios 50
7.11 Convergence of proposed algorithm while self-monitoring T_{off}, with different number of pseudonym links 51
7.12 Distribution of estimated mean over own value, 25 pseudonym links, availability 20% 51
7.13 Distribution of estimated mean over own value, 75 pseudonym links, availability 20% 52
7.14 Convergence when using churn from Skype traces presented in [31] 53
7.15 Mean T_{off} distribution and estimate distribution with Skype traces presented in [31] 53
7.16 Proposed algorithm, mean estimate and standard deviation over time 56
7.17 Anonymity set size 57
7.18 Mean estimate and standard deviation and anonymity set size of Skype traces presented in [31] 57

List of Tables

6.1 Definitions used for describing the algorithm 31
6.2 Properties of a node 31
6.3 Properties of a pseudonym 31

7.1 Default values for parameters in proposed solution 42
7.2 Standard deviation of estimates among online nodes in proposed algorithm 48
7.3 Standard deviation of estimate among online nodes with different number of pseudonym links pr node 52

Part I

Introduction, background and application context

Chapter 1

Introduction

Online social networks (OSN) such as Facebook and Twitter have revolutionized how most people use the internet. Hundred of millions of users are generating staggering volumes of data, which get stored in the servers of the operators. In 2012 Facebook stored over 500 Terrabyte of new data daily. At the same time twitter was receiving over 340 million tweets a month. Since then both companies have continued to grow both in terms of users, and data generated. In short, we as users, are trusting a huge amount of personal data to the same domain, making them both more valuable, and more vulnerable for attack. The companies themselves make their lively hood of this data, earning money by offering targeted marketing and selling statistics about their users and their behaviour to other companies. Intelligence and law enforcement agencies around the world are routinely monitoring OSNs finding treasure troves of information. We are trusting the companies behind the OSNs with a lot, and maybe to much, information without really knowing how our information is being used, by both the companies themselves or by others. Not to mention how criminals could exploit the information if they where to get hold of it through malicious attacks or other schemes.

The advantages of users having more control with how their personal data is being used should be obvious. One of the best ways of ensuring this would be to give the user the ability to control who should have access to which information, while nobody else should be aware of the existence and/or the origin of the information, unless disclosed by one of participating parties (sender or receiver). To the best of our knowledge, there are currently no existing large scale OSNs which offers these options to their users. Not to mention that with a centralized infrastructure, this would also be hard to realize as the centralised entity would see at least the communication patterns. However, as proposed by [33] friend-to-friend networks (described in Section 2.2) such as Freenet have a good potential to offer users such control.

The protocol presented in [33] is a promising start for a system which could offer robust privacy-preserving data dissemination over a social graph. It does however require a method for estimating churn in a privacy preserving manner. Estimation and aggregation in P2P networks is in itself

a challenging task, as there are no centralized component which can gather information from the nodes. The authors of [15] and [13] have proposed solutions for aggregation in P2P networks, but common for most works in the area is that performance suffers in the presence of churn. This is of course a crucial factor when one wants to estimate churn. There are existing works which do estimate churn in P2P networks, such as [25] and [2], but these do not consider estimation in a privacy-preserving manner.

To the best of our knowledge privacy-preserving estimation of churn in large scale P2P networks has yet to be considered by literature. Every standalone component such as aggregation, churn estimation and privacy in P2P networks has been considered, but never together. This is our main contribution with this thesis. We propose a solution for robust privacy-preserving estimation of churn in large-scale P2P networks.

The privacy-preserving capabilities in [33] come from the use of pseudonyms with limited lifetime. Nodes will periodically create new pseudonyms for themselves invalidating old pseudonyms. And unless there is a trust relation between two nodes, the only way of contacting a other node is via a valid pseudonym. This enables the creation of a robust overlay network for data dissemination. Our contribution in this context is that we offer a solution for churn estimation which makes it virtually impossible to track individual nodes across multiple pseudonyms, and by doing so maintain the privacy-preserving capabilities of [33]. Our work should also lend it self to do privacy-preserving estimation in other contexts than [33], assuming that there is some mechanism for hiding individual nodes (like a pseudonym) present.

The gist of our solution is as follows. Every node will keep a set of estimates from other nodes in the network, including offline nodes. Based on these estimates and observations about itself it will periodically make an estimate about the churn in the network and gossip this estimates to the other nodes so that its estimate can be included into the set of other nodes. The estimates are calculated as the mean of all observations in the set and the self observation. By doing this all estimates will converge toward the same number resulting in only a small deviation among the estimates. This is crucial as it is this small deviation which forms the basis for privacy-preserving capabilities of the protocol. By having all nodes gossip approximately the same values tracking individual nodes across pseudonyms becomes virtually impossible. The protocol also allows for changes in the churn pattern without restarting the protocol as nodes periodically make a new estimate based on both self observation and estimates from other nodes and then disseminate these changes to their neighbours. This dissemination is also very robust as it follows the same principles used to create the robust network overlay.

We also introduce, to the best of our knowledge, a new metric for evaluating the degree of privacy an anonymity set can provide based on the probability that two nodes will transmit an indistinguishable value. The authors of [32] and [8] identified the need for an additional metric for evaluating the degree of anonymity provided by anonymity sets beyond the set size. The set size should be coupled with a threshold

defining whether a entity is within the set or not. However, to the best of our knowledge there is no literature which provides a generic way of calculating this threshold for numbers. Having such a metric for anonymity sets would enable a better basis for comparison of the privacy-preserving capabilities of different estimation algorithms.

In Chapter 7 we show that our solution is capable of estimating the mean churn in the network with reasonable accuracy and within a reasonable amount of time. This holds true in a range of artificially generated churn scenarios with varying node availability, as well as under real-life churn using churn traces from [31].

We also show that the protocol proposed by [33] has similar performance when running our churn estimation protocol than it did when having perfect knowledge about the churn. The evaluation of the privacy-preserving capabilities of our protocol by our new metric also shows satisfactory results.

Finally we identify some properties with our proposed solution which might deserve some future attention if one wishes to use our solution in a real-life scenario.

Thesis structure

The rest of this thesis is structured as follows: We first describe some of the key concepts and terminology used in this thesis in Chapter 2. This is followed by the related work in Chapter 3. In Chapter 4 we describe the application context in which the work in this thesis is done, before we describe the problem we wish to solve in Chapter 5. Our proposed solution is presented in Chapter 6 and evaluated according to our performance metrics both of which are found in Chapter 7. Finally we draw our conclusions and present our ideas for future work in Chapter 8.

Chapter 2

Background

In this section we explain and describe some of the terminology and key concepts which we use either for explaining or evaluating the work presented in this thesis. These are brief introductions and are meant as a complete explanation of the topics, but they are meant to give sufficient information to understand what is being presented.

2.1 Peer to peer networks

Peer to peer (P2P) networks are a class of decentralized distributed networks where nodes (peers) supply and consume resources. P2P networks distinguish themselves from more traditional layer-3 networks in many ways. They tend to be highly dynamic, having a high join and leave rate, as peers periodically come online and go offline. They usually do not have the hierarchical topology of traditional networks, which typically comes form how devices are physically connected to each other (and by which role these devices have).

Instead a P2P networks are logical networks, where the topology of how peers are connected to each other is determined by some logical or random factor. P2P networks also tend to be either partially or fully decentralized networks where the network does not rely on a known ever-present entity (such as a internet server). Or as in the case of a partially decentralised network relies on the centralized component as little as possible and typically only in the start up/join phase. Another distinguishing feature of P2P networks, is that peers are generally speaking treated as equals. This means that they are operating both as servers and clients towards the other nodes, and are expected to be able to take different roles in the network as the situation requires. By comparison, in more traditional networks participants usually have clearly defined roles in terms of who is the server and who is the client, not to mention which role they have in the topology.

It is worth noting that P2P networks still require some physical medium to communicate with each other. This will in most cases be a traditional hierarchical network (for instance the internet) or some sort of wireless sensor network. When talking about protocols running in P2P networks

or P2P networks in general in this thesis, we assume that the underlying communication is present and that the P2P network acts as middle-ware between the transport layer and the application layer of the TCP/IP stack.

The lack of (or limited use of) a centralized component is one of the features which makes P2P networks attractive from a connectivity perspective. If one does not rely on a central structure (such as a server) one is avoiding single points of failure. The drawback of not having such a centralized structure is that joining a P2P network in the first place becomes difficult for peers. To join a P2P system they would have to know the address (IP or similar) of at least one of the peers which is already connected to the network, and at least one of these peers needs to be online at the time of connection. This is the reason why most P2P networks that exist today have some centralized component which helps nodes which want to join the P2P network to find other peers which are online at the present time (bit-torrent trackers are an example of a well know centralized component).

Continuing the train of thought of wanting to avoid relying on a centralized component, creating and maintaining an efficient network topology for the P2P network becomes a task the peers have to share between themselevs. Literature has yet to come up with a definitive answer to how this should be done, and there have been, and still are, a multitude of techniques proposed for the best way of doing this. Classifying different techniques for creating and maintaining network topology in P2P networks would be a mammoth task, but solutions generally fall into one of two main categories in regard to network topology: structured and unstructured networks.

2.1.1 Structured P2P networks

Structured P2P networks try, as the name implies, to create a specific structure as a network overlay. The general idea is to create some form of hierarchy that allows for efficient routing between the peers. The creation of a hierarchy allows nodes to determine their role in the network and ideally creates an even work load for all the participating peers. Maintaining the hierarchy can require a lot of additional computations and/or messages, especially if peers leave and join a lot, however it brings with it the advantages of a deterministic scheme to do so. A deterministic scheme also makes it easier to provide guarantees in form of guaranteed message delivery to all peers etc. Node placement in the hierarchy is often determined by peer ID or some proximity metric between nodes. Distributed Hash Tables, like described in [1], is a typical example of structured P2P networks.

2.1.2 Unstructured P2P networks

By contrast the unstructured P2P networks do not follow a predefined scheme for crating the network overlay. Instead, links are most commonly created in a more random fashion, as for instance in Gnutella. Unstructured

P2P networks typically have the advantage of spending less computations and messages to maintain the network overlay. This is an advantage for large networks, and networks which are highly dynamic or networks which are both large and highly dynamic. They are often designed to support rapid information dissemination, however give less or no guarantees about all information reaching all peers. This means that unstructured P2P networks often requires additional mechanisms if high probability for reaching all nodes with information is required.

2.2 Friend to friend networks

Friend to friend (F2F) networks are a type of P2P network where participants can only make direct contact with peers with whom they are friends. The topology of the network is equal that of the social graph, having links between the nodes only where there exists a relation of friendship. The general idea is that if you only communicate with people you are friends with and trust, you are not disclosing information about yourself or your resources to outsiders, who you might not trust. Information can be disseminated through the network and to nodes which are not friends by having common friends relaying messages. This means that if a node wishes to disseminate a message to everybody in the network, it will send the message to all its friends, and then have all friends forward the message to all their friends, and so on.

Another important feature is that if a node is relaying a message from one friend to another, it does not know (nor should it) if the friend sending the message is the original sender of the message, nor if the friend to which the message is forwarded is the final recipient. So in essence the purpose of a F2F network is to create a P2P network where it is hard or impossible to track who has which resources, and who is spending which resources. Freenet [7] is a good example of a F2F network.

2.3 Gossiping

Gossiping, also known as epidemic dissemination, is a widely used scheme for disseminating information in P2P networks. Node a can tell node b a piece of information which node b in turn can spread to all its neighbours in the network which in turn can tell their neighbours. It can be compared to infecting nodes with information which in turn infect their neighbours, like a virus would do, hence the term epidemic dissemination.

2.4 Churn and models for churn

Churn which is the short form of churn rate, is a metric used to describe how many nodes leave or join a network over a period of time. If a network has a high churn rate (or high churn) the number of nodes that leave and join the network is high, usually indicating that nodes leave frequently,

either to rejoin the network at a later point or to be replaces by new nodes. A network with a low churn rate will usually have nodes which are online for a significant amount of time and seldom or never leave the network.

There are currently two main ways of defining the churn rate in P2P networks. The first is to consider churn as the number of nodes which leave or join the network in a given time interval. This can be interesting knowledge if one for instance has to adapt the overlay to include new nodes and remove nodes which have left. This is a good way of measuring churn if one assumes that a lot of nodes will not return to the network, or return so infrequently, that it does not pay to keep track of them. Methods for calculating churn in such networks have been proposed in for instance [9] and [11].

The other main way of seeing churn, and the one which we are using in this thesis, is to see how long nodes remain connected to, and disconnected from, the network. The basic assumption is that nodes will at some point, within a reasonable amount of time, rejoin the network because they have strong incentives for doing so (resource sharing etc). This knowledge can then be used to for instance make better decisions about how to create the network overlay. A nodes online time is usually described as its *Ton* time and its offline time *Toff* time.

2.5 Random graph

A random graph is a graph that can be created by taking a set of isolated nodes and then randomly adding edges between them. In many systems described in literature, as well as in this thesis, authors try to achieve a overlay that resembles a random graph. The reason for trying to achieve a random graph is that random graphs have some desirable characteristics which we are interested in when designing an unstructured overlay network.

The probability that a random graph remains connected even though you remove a high number of edges (nodes that have left the system/are offline) is very high, given that each node has sufficient edges. This means that we will have a connected overlay even though the churn in the network is high. Another desirable property of a random graph is that it scales well, or to be precise, you can double the number of nodes without having to increase the out-degree of individual nodes by much, and still maintain the same probabilities for connectivity.

This scalability in number of nodes without increasing system complexity is also one of the strong suits of many unstructured overlay networks and is one of the reasons why random graphs lend themselves particularly well to these kind of networks. One of the more comprehensive works on random graphs and their properties is [3], and more details and reasoning about random graphs can be found there.

2.6 Standard deviation

Standard deviation is a statistical property which shows how much numerical variation there exists from the mean in a set of numbers. A low standard deviation means that most values are close to the mean, while a high standard deviation means that the numbers in the set are spread over a large range of values. Say that the mean (average) height of all men in Norway is 180 cm. If the standard deviation is 3 (assuming normal distribution), this means that 68% of all Norwegian men have a height within +/- 3 cm of 180 cm (they are between 177 and 183 cm high), and 95% of all Norwegian men will have a height within +/- 6 cm (2x standard deviation) of 180 cm.

2.7 Cumulative distribution function

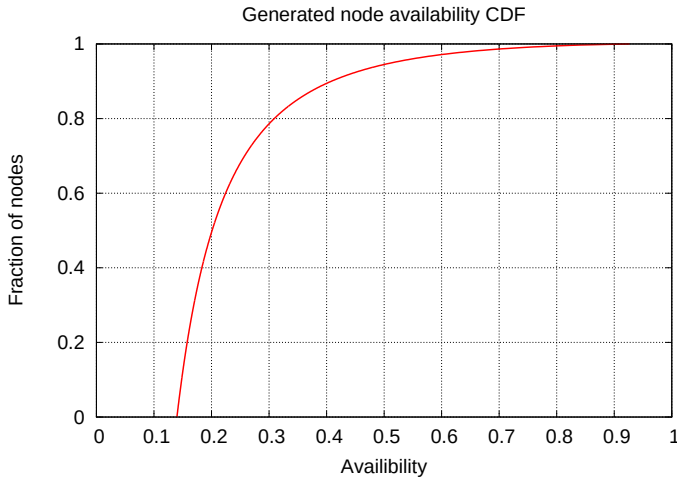


Figure 2.1: CDF example: CDF distribution of availability

Cumulative distribution function, or CDF for short, is a statistical property which describes the probability that a random variable x can with a given probability be found at a value equal to, or less than, y. Figure 2.1 is an example of how a CDF distribution can be visualized. We are showing the CDF distribution for node availability which has a mean availability of 20%. We can read from the Figure that the probability that a randomly selected node has an availability equal to, or less than, 0.3 is about 80%. This in turn tells us that the majority of nodes for this availability distribution have relatively low availability. A plot of CDF gives an accurate visual description of how the values in a set are distributed.

Chapter 3

Related work

In this chapter we present some of the recent literature which is relevant for this thesis, and present our views on their strengths and weaknesses.

3.1 Distributed aggregation

All good estimates are based on relevant information. The more relevant information one can assemble the better an estimate can potentially be. To assemble relevant information we aggregate data.

Aggregation in large-scale dynamic distributed systems, such as P2P systems, has been well studied in the past. Solutions typically fall into one of two main categories, hierarchical or gossip based solutions. The hierarchical approaches such as [10, 16, 34, 37] are distinguishable by nodes getting organized in tree-like structures and messages getting passed up and down the tree structure. This allows for high accuracy in aggregation with relatively low cost in terms of workload and message-passing once the tree-like structure is created, but requires a lot of messages to maintain the structure in the presence of node churn.

The gossip based approach, such as [13–15, 23, 30], generally speaking rely on exchanging information with randomly selected neighbours. This probabilistic approach is more scalable than the hierarchical approach as it does not require the creation of tree-like structures, but is less accurate as it is probabilistic. In this thesis we will focus on the gossip based approaches because we wish to do estimation in an unstructured P2P network.

Finding or creating an good aggregation technique is crucial for finding a good solution for churn estimation. In the following section we are taking a closer look at some aggregation techniques which potentially could be used as the basis for our own aggregation and point out why we like them and some problems which have to be overcome if we wish to use them in our estimation.

3.1.1 Gossip-based aggregation

Gossip based aggregation of global parameters in P2P networks such as *avg*, *sum*, *min*, *max* is a problem which has been studied in many scenarios.

One of the most referenced works in the area is [15]. The idea is simple but effective. Whenever two nodes gossip, node a will send its value Xa to node b which in return sends its value Xb to a. The nodes then update their value to be $(Xa + Xb)/2$ (mean), min/max(Xa, Xb) or something similar for desired effect. The effect of min/max is obvious. In the case of mean, the idea is that with every gossip interaction the values held by individual nodes becomes less extreme until finally converging on the mean for the entire system.

It is worth noting that how fast the system will converge is dependent on how the nodes choose to gossip. Choosing nodes in a fashion which ensures short path lengths in the graph is generally better than choosing gossiping partners which create long path lengths (at least if we disregard constructed scenarios where nodes will choose gossiping partners based on how far away they are from the mean).

The main problem of the algorithm, especially for aggregating the mean, is churn. If a node for some reason leaves the network before the system has converged, the system can not converge to the correct value (which is the mean of all nodes online at the beginning of the run). The earlier a node leaves the more off the result will be. This because the value held by the node has been considered less than others. Equally, if nodes would be allowed to join the averaging while in progress the system would never converge (as long as new nodes are being added), nor would it converge to the correct value (unless by chance).

To address these issues one would typically run multiple parallel instances of the algorithm and compare results between runs, which can compensate for nodes leaving, while prohibiting nodes from joining instances which started before they joined the network. However even this would not guarantee a convergence in networks with high churn as the chance of a significant amount of nodes leaving the network early in a aggregation cycle is high, skewing the results.

Other proposed solutions worth mentioning are [12] and [17]. They have commonalities with [15] but offer better domain specific solutions for aggregation in terms of either message overhead or convergence. They do however suffer from the same disadvantages in terms of behaviour under churn and need to periodically restart as [15] does.

Another interesting approach to aggregation is the approach presented in [13]. The basic idea is that each node is maintaining a sample of values. During each round of gossiping nodes exchange samples and then updating their own selection of samples with values from the gossiping. Exploring some different scenarios for merging samples, ranging from random replacement to different schemes for merging with the goal of storing the maximum amount of information with the minimum amount of space required they found some interesting properties. The clever approaches which captured more data, where more accurate than the simpler approaches such as random replacement. They also determined that the presence of duplicates (that a value from a single node gets included multiple times into the same sample) has a relatively small effect on accuracy. Thus one could potentially neglect to track which nodes

are already included into samples allowing for simpler algorithms. The main advantages with the technique presented is that one does not need to keep track of individual nodes when selecting samples, and if using clever techniques for merging samples one can achieve quite accurate results with minimum message overhead. The main drawbacks are that the protocol would have to be restarted at certain time intervals to get updated information, and that there is no data on how this system would preform under the presence of churn.

3.2 Churn estimation

The effects of churn on P2P networks have been studied in many scenarios such as in [18–20], and there has been quite a few studies on how to minimize the impact of churn on P2P networks [21, 26, 29]. However, to the best of our knowledge there has not been so many studies focusing on how to estimate the amount of churn in a P2P network, though there are some [2, 9, 11, 25].

Understanding how churn impacts a P2P systems is important as many aggregation techniques are also suffering in performance under the presence of churn. This is also why estimating churn is a potentially even more challenging task than estimating other properties in a P2P system. In the following sections we take a closer look at a few of the recently proposed solutions for estimating churn in P2P networks. We are again focusing on analysing their strong-suits and weaknesses looking for possible solutions to our problem, or issues which needs to be resolved to create a working solution.

3.2.1 Estimating leaves and joins

As described in Section 2.4 one of the two main ways of modelling churn is to see how many nodes leave and join the network within a certain period of time. In [9], and similarly in [11], the authors propose a way of doing just this. In stead of counting the exact number of leaves and joins in the network, they instead propose to monitor the join and departure rate relative to the size of the network. The join/departure rate for a given time period l would be the number of joined/departed nodes within that time period divided by the number of nodes in the network. This is done in the following manner: Given an arbitrary P2P network where every node maintains a fixed number of overlay neighbours C, every node which does not leave or join the network within the fixed time period l, monitors and records all nodes departing or joining the network. Joining nodes are monitored by requiring every node to send a special JOIN message to an online node at the time of joining (or rejoining) the network. A node can detect departing nodes by monitoring the neighbouring nodes in C. Once the period l is over, all nodes which did not depart or join within l calculate the relative join and departure rate as $joins/C$ and $departures/C$. The nodes then average these values across the network giving the relative

global join and departure rate. The neat thing about this approach is that it will give a fairly accurate estimate of the relative global leave and join rate within a certain time interval, without prior knowledge of the size of the network. This is important as estimating the size of the network is in it self an aggregation challenge in P2P networks. The approach also has the advantage of working regardless of how the underlying network overlay is created, as long as how often each individual node appears in the overlay links of the other nodes is not too skewed. The main drawback is that to work, the system requires some sort of synchronisation mechanism between nodes so that they can agree on when and how long individual instances of l are. To always have access to up to date information one would also have to run multiple parallel and continuous instances of the protocol.

3.2.2 Estimating T_{on} and T_{off}

A solution for estimating churn in terms of T_{on} and T_{off} in structured P2P networks is proposed in [2]. The idea is that each node will maintain a set of observations from a selection of nodes, creating a sample set of observations, and then making its estimate based on this set of observations. The observations can either come from the nodes observing their neighbours in the overlay, or by nodes reporting observations about themselves to their neighbours. In either case a node will, given time, get a set of observations about its neighbours. The bigger this set is, the more accurate the estimation can be, as it is based on more information. However, a bigger set will take more time to collect and will potentially contain more stale information if one does not take steps to ensure freshness. By stale information we in this case mean observations which are so old that they are no longer relevant. Thus the set size becomes a trade-off between accuracy and message complexity.

The approach has the benefit of working with a wide range of different overlays. In [2] they use a structured overlay network, however there is to the best of our knowledge nothing that prevents this approach from being used in other forms of overlay networks. Though there has to be some mechanism for selecting sample sets in such a way that each node appears with approximately the same frequency across all sets.

As pointed out in [2], when a node estimates the system wide mean T_{on} or T_{off} by calculating the mean of the observations in its set it does not guarantee a correct estimate for all nodes. However the mean of all estimates should be the same as (or close to) the mean of all individual observations. And the bigger the observation sets are, the smaller the deviation among the estimates should be. Another advantage of the algorithm is that it will adapt to changes in churn without running multiple instances, as the observations can be updated over time. It also does not require any special synchronisation mechanisms unless the observation metric requires this. The main drawback of the proposed solution is that all nodes do not estimate the same correct value, which for some systems

might be crucial.

3.3 Evaluating privacy in networks

Our goal is to come up with a method for estimating churn in a privacy-preserving manner. This presents an interesting conundrum. Estimation relies on aggregation which means that information needs to be disclosed and exchanged. Disclosing information about oneself is not something one would usually do when trying to preserve privacy. Hence we need a way for nodes to disclose information which does not compromise privacy.

Privacy and anonymity are two closely related terms. One being slightly weaker than the other. If nodes can remain anonymous they are also privacy-preserving, but preserving privacy is not equivalent to being anonymous. Hence, we are aiming for anonymity, but can settle for privacy-preserving. Our notion of privacy is further described in Chapter 7.4.

When trying to achieve anonymous communication on the internet there are mainly two approaches one can take. You can either use cryptography like proposed in for instance [6] or try to blend into the crowd making observers uncertain of your actions like proposed in [5]. Cryptography can be an effective approach when one wishes to hide what one is communicating (for example VPN), but does in it self not hide who you are communicating with. This makes applying cryptography to hide the identity of a node a less likely approach to succeed. We need the information which is transmitted, but wish to hide who is sending it. This leads us to the other main approach for achieving anonymous communications. Trying to hide who is communicating with whom is typically done by gathering a mass of users into a crowd, where tracking individuals gets almost impossible.

In [5] they introduced the term *anonymity set* as a metric for the amount of anonymity provided in such a crowd. The idea being that if an individual in the set sent a message, nobody inside or outside the set should know the message's origin for certain, except the actual sender. Thus staying anonymous comes down to probabilities and a large anonymity set is the goal.

Over the years there have been several techniques proposed for creating an anonymous internet experience for internet users. Both [27] and [28] propose possible approaches for users to create a large anonymity sets for themselves, and in recent years the P2P based **TOR** network [22] has become a working practical solution for being virtually untraceable on the internet.

The aim of having an anonymity set which is as big as possible is a good principle, but has some issues which should to be considered. The main issue is that there to the best of our knowledge does not exist any strict definitions on how to determine which nodes qualify to be in an anonymity set. In theory any node which has the potential of being mistaken for the node which is trying to hide could be considered as a part of the anonymity

set of that node, there is however one major problem with this definition. Think of the following scenario. Say you and 10 friends are running a mixing protocol which is poorly designed, and you send a message where you do not wish to be identified as the sender. An external observer is observing outgoing messages from your mixing pool, but is still able to estimate which messages originally come from you with a very high probability due to the poor design. Your friends are technically still a part of your anonymity set as there still remains some slight chance that they where the originators of the message, but for all practical purposes they should probably not be included in your anonymity set as the probability for you being the originator is so high. This weakness in definition was pointed out by the authors of both [32] and [8], and hence we need a stricter definition for our anonymity sets.

In [24] they define anonymity as: *"Anonymity is the state of being not identifiable within a set of subjects, the **anonymity set**"*. Using this definition the authors of [32] and [8] identified that it is important to consider the probability an attacker would assign to each individual entity in the anonymity set for being the originator of a message. In particular [8] also identified that within an anonymity set each entity should ideally have the same probability for being the originator to provide maximum anonymity. Hence both probability of identification and set size should be taken into consideration when evaluating an anonymity set. And a *strong anonymity set* provides many nodes which all would be assigned a high probability of being mistakenly identified as the node for which the anonymity set is being provided. To the best of our knowledge there does not exist any definition for how the strength of an anonymity set should calculated when considering numerical data values transmitted over time.

Chapter 4

Application context

We wish to build the basis for a robust privacy-preserving protocol for data dissemination in a delay-tolerant network. Such a protocol could in turn be used to build for instance an OSN which offers the users more control and privacy from both internal and external observers, as well as threat from leaks (malicious or otherwise), than current centralized solution do. In this Chapter we present the proposed solution for creating such a system as presented by [33]. This will be the application context for this thesis.

4.1 A robust privacy-preserving protocol for data dissemination

As pointed out in [33] F2F networks form a good basis for designing privacy-preserving protocols because they provide a lot of user privacy. Friends can only communicate with other friends so if a node adds new friends to the network this is of little concern for other nodes as long as they trust their friends not to disclose information about them. This is because additional friends of node a will have no direct way of contacting node b by any other means than having a common friend relaying the message. Equally, information originating from node a can not be seen by other than friends of a unless one of them breaks its trust and forwards the information to their friends. Information can be forwarded but the originator can not be found or tracked unless a node which has a trust relation with the originator discloses its identity. This makes F2F networks highly scalable. On the other hand, one of the biggest drawbacks with a F2F network is that they typically do not have a layout which allows for efficient disseminating of messages across the network, as most other P2P networks do. Neither are F2F networks, which typically resemble a social graph, as strong in terms of connectivity as other P2P networks typically are, and could easily become disconnected even under a modest churn. Hence the idea of [33] is that to create a protocol which takes a social graph (F2F network) and supplements it with additional links to give it the robustness of a P2P network with an overlay resembling a random graph. The desired effect is illustrated in Figure 4.1. The problem with

designing such a system from that the additional links which have to be created to supplement the social graph, are between nodes which do not trust each other. This means that they have to be made in a manner which does not disclose node identities while still providing an abstraction for robust privacy-preserving routing. By combining the robustness from the P2P world with the privacy-preserving capabilities of the F2F network [33] proposes to create an efficient, scalable, reliable and privacy-preserving protocol which can be used for data dissemination.

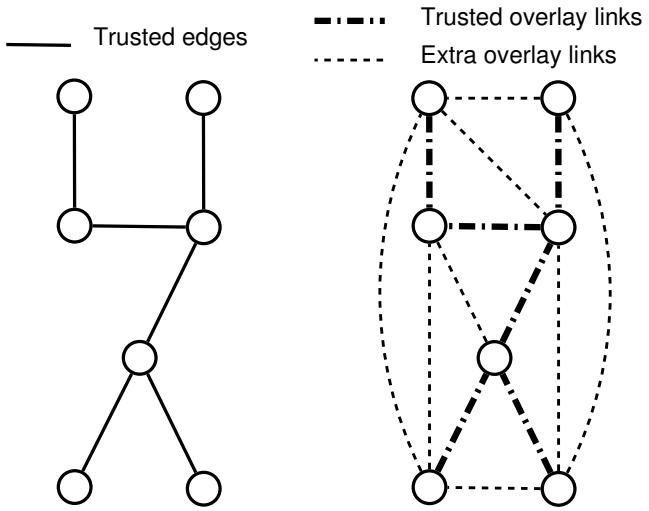


Figure 4.1: Example of a trust graph and derived communication overlay presented in [33]

4.2 Privacy guarantees

Under the assumption that no participating node will intentionally disclose its own participation or the participation of any of its trusted peers in the network, [33] defines a privacy-preserving system which provides the following guarantees.

First and foremost, nobody outside the network is able to determine who is participating in the network nor are they able to join the network without establishing a trust relationship with at least one node already inside the network. The only information a node has about the network is its own list of trusted peers.

Secondly the system does not disclose the edges in the trust graph to an entity monitoring the system with a high probability. If node a trusts node b and node c, node a is unable to determine whether or not node b trusts node c.

Third, the data disseminated through the network is readable only to the members of the network.

4.3 Overview of the protocol

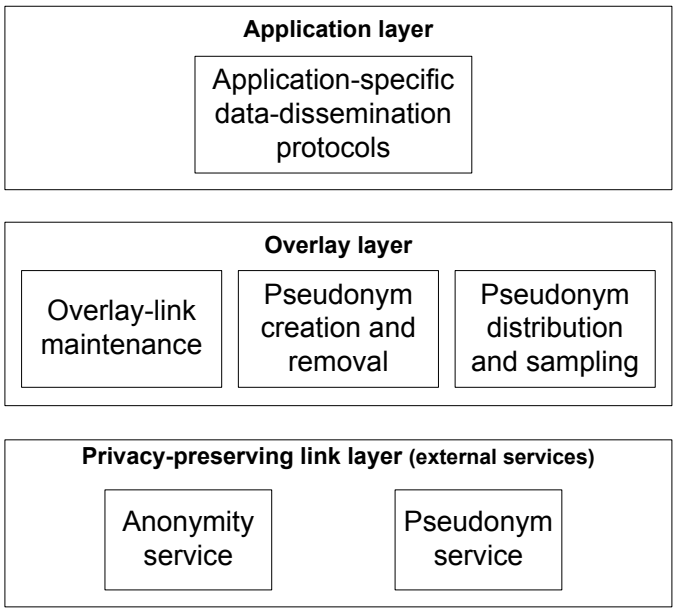


Figure 4.2: Architecture for privacy-preserving data dissemination presented in [33]

To create a protocol which would satisfy the criteria described in Section 4.1, the authors of [33] propose a three layered approach (Figure 4.2), where the lowest layer (*privacy-preserving link layer*) consists of an anonymity service and a pseudonym service. The *anonymity service* is responsible for providing privacy-preserving end to end communication between two nodes with known IDs (*trusted links*). The *pseudonym service* on the other hand is there to provide privacy-preserving end to end communication to a node whose pseudonym is known by the sending node (*pseudonym links*). These services have some additional requirements which are not relevant for the working of the algorithm but are related to the definition of privacy Section 4.2. It is also worth noting that how these services should be realized falls outside the scope of both [33] and this work.

The next layer, called the *overlay layer*, is responsible for creating and maintaining the network overlay. *Overlay links* are considered the union between *trusted links* and *pseudonym links*. When a node rejoins the system it will re-establish the overlay links it had before leaving. In the same way links to nodes which have become offline are not removed from the overlay and will again be operational if the node rejoins the system. Hence, the system does not provide guarantees for individual links but gets its robustness from redundancy given by the collection of links.

Trusted links are considered as static links for the purposes of this work, and maintaining them is hence a straight forward operation. Nodes joining the system for the first time will not have any pseudonym links, but will have to rely on the trusted links as its initial overlay. To create, spread and

remove pseudonym links the overlay layer runs a maintenance protocol which has schemes for creating, removing and propagating pseudonyms.

Pseudonyms are created with a limited lifetime to enhance privacy. The scheme for creating and removing pseudonyms is described in further detail in Section 4.3.1. Pseudonyms are disseminated throughout the overlay using a gossiping algorithm, and each individual node runs a sampling algorithm across the gossiped pseudonyms to select which pseudonyms should be in its *overlay links* with the goal of having an overlay that resembles a random graph. The gossiping is described in Section 4.3.2 and the sampling in Section 4.3.3.

The *overlay layer* is also responsible for providing higher layers a way to communicate with the nodes in the overlay, but how this is done falls outside the scope of this work.

4.3.1 Creating and removing pseudonyms

The first time a node joins the system it creates a pseudonym to represent itself. Pseudonyms always have a limited lifetime to enhance privacy (see Chapter 5 for more details), and any pseudonym that expires will become invalid and is removed from the system. When a node's pseudonym expires it will create a new pseudonym for itself if it is online (or it will create one once it again joins the network). The pseudonym service guarantees that pseudonyms remain valid even though the node that created it goes offline. It also ensures that a node can be contacted again through the service once it rejoins the system given that the pseudonym has not expired.

4.3.2 Gossiping pseudonyms

Gossiping is done using the shuffling idea presented in [35]. Every node maintains a *pseudonym cache* of configurable size n. In the beginning the cache is empty. Periodically every node selects a uniformly random link from its *overlay links* or *pseudonym cache*. The node a then executes the shuffling protocol with the selected node b as the target. a and b then exchange messages with each other containing a sample of up to l (which is also configurable) number of pseudonyms. The sample will include up to l-1 pseudonyms from the nodes cache, and the nodes own pseudonym (less if there are not sufficient pseudonyms in the cache). The replacement policy is also similar to the one described in [35]. Upon receiving a number of pseudonyms a node will first fill the cache. If the cache is full a node will then replace the pseudonyms which it just sent to its gossiping partner with the newly received pseudonyms, while removing any duplicates or occurrences of its own pseudonym. This means that with every shuffle a node will, with high likelihood, replace at least some of its cache with new pseudonyms ensuring dissemination of the pseudonyms. In addition to applying the sample to the cache, nodes will run the sampling mechanism described in Section 4.3.3 whenever it shuffles.

4.3.3 Selecting pseudonym links

Sample selection for the *overlay links* is done by the principles outlined by [4]. Every node keeps configurable number of links s which make out the *pseudonym links*. The bigger s is the more robust the overlay will be. However, every additional link creates more computational complexity to maintain. In addition we keep a small number of potential candidates for each pseudonym link q. The goal is to create a network overlay which resembles a random graph (which is good for robustness) which allows for fast pseudonym link replacement when a pseudonym expires. This is done in the following manner:

To each pseudonym link slot we attach a uniformly random number, and either a pseudonym or an empty value. Every pseudonym has a uniformly random number attached to it. Whenever a node receives a pseudonym, for all pseudonyms that the node currently knows of, it considers the following when deciding which pseudonyms should be a part of the pseudonym links:

1. If a pseudonym has expired, discard it.
2. If a pseudonym link has an empty value attached to it, replace the empty value with the pseudonym.
3. If a pseudonym's value is numerically closer to the value attached to the pseudonym link than to the value held by the current pseudonym, replace the existing pseudonym with the new one. The former holder becomes the head of the queue for potential candidates of that pseudonym link, booting the last one if the queue is full.
4. If a pseudonym's value is not closer to the value attached to the pseudonym link than the current pseudonym, consider it for a slot among the potential candidates to this pseudonym link. Candidates are ranked by numerical closeness.
5. In case of a tie between two or more pseudonyms for a link overlay slot or a potential candidate slot, the pseudonym with the longest remaining pseudonym lifetime is chosen first.

Note that pseudonyms can appear multiple times in the overlay, both as pseudonym links or as potential candidates for one.

Part II

Problem statement, solution and evaluation

Chapter 5

Problem statement

In [33], which is summarized in Chapter 4, they proposed a system for robust privacy-preserving data dissemination. They did however make multiple assumptions. In this thesis we are addressing how nodes can be able to correctly estimate the amount of churn in the system. In short, we are trying to solve how to estimate mean *Toff* in the system without compromising node privacy.

5.1 Pseudonym lifetime and privacy

Privacy in [33] is mainly achieved by use of pseudonyms. Pseudonyms are a good way of staying concealed, however they become easier to debunk the longer they last, so short lifespans are preferably. The drawback of having short pseudonym lifespans is that the system uses the pseudonym links to create and maintain a robust network overlay. Changing pseudonyms to often would result in a larger message overhead to maintain the overlay and would at some point result in a less robust network. Experiments shown in [33] showed that a pseudonym lifetime of three times the mean *Toff* time in the system was a good pseudonym lifetime as far as connectivity is concerned, and no significant improvements in connectivity was achieved by increasing the pseudonym lifetime. If we assume that a pseudonym lifetime of three times the mean *Toff* time in the system also is an acceptable value for privacy preservation we must conclude that making a system as proposed in [33] is possible as long as we can estimate the mean *Toff* in the system in a privacy preserving manner.

5.2 Estimating mean T_{off} time in a privacy-preserving manner

As privacy-preservations is a key requirement for the existing protocol, the estimations should be done in such a way that they do not disclose information that would compromise node privacy. From a privacy-preservation point of view, dissemination of observations should preferably only go

through the trust graph. This would be equal to do dissemination in a F2F network and we could use any aggregation algorithm without compromising privacy. The drawbacks of disseminating only over the trust graph is that F2F networks typically have longer path-lengths than other P2P networks, and are highly vulnerable to churn. Hence in terms of convergence speed we would preferably do any aggregation over a random graph which is robust to churn and has short path-lengths (as we do not have a structured overlay), meaning that aggregation should preferably be done with the help of the privacy-preserving overlay links. This would enable us to for instance use approach presented in [2]. However, doing so would disclose information over the untrusted links which could harm privacy as one could track nodes across pseudonyms through the values they report. Hence we need a protocol which can communicate over the privacy-preserving links without disclosing information which can be used to track the identity of individual nodes across pseudonyms and hence compromise the privacy-preserving capabilities of the existing protocol. Our notion of privacy-preserving is described in Section 4.2.

5.2.1 Formal problem statement

The problem that need to be solved is as follows: Given a protocol which allows for robust privacy-preserving for data dissemination, create a protocol which allows for a reasonable accurate estimation of the mean T_{off} times across all nodes in network, while satisfying the following main criteria:

- *Privacy-preserving estimation:* Estimating churn should not disclose the identities of nodes, nor the relations between them.
- *Robustness:* The protocol should be able to give accurate estimates under the presence of realistic churn.
- *Has fast convergence:* A protocol that does not converge in reasonable time is not particularly useful.

Chapter 6

Solution

We are trying to solve the problem of estimating mean *Toff* time in a privacy preserving manner with the work presented in [33] as our application context. In this chapter we first discuss which considerations have to be made when designing a protocol for this context in Section 6.1, followed by which assumptions where made when designing the protocol in Section 6.2. We then go on to present our solution in Section 6.3 followed by the reasoning behind our choices in Section 6.4. In Section 6.6 we elaborate on how our approach could be applied in other contexts and which considerations have to be taken if doing so. We finally make a brief summary of our solution in Section 6.7.

6.1 Design considerations

There are some key issues that need to be addressed when trying to estimate churn in a P2P network. Like all estimates we want as much relevant data as possible to make a good estimate, which means that we need to aggregate data from nodes across the network. Aggregating data in P2P networks is not a new problem, and as presented in Section 3.1 there are multiple possible solutions.

One of the main properties which makes estimating churn different than estimating other properties in a P2P network is that we are also interested in the properties from nodes which are not currently online. This is especially true when churn is modelled as *Ton* and *Toff* times as we want actual values, not the leave/join rate as in [25]. In [2] estimating churn as *Ton*/*Toff* is being solved by considering values from a selection of currently online nodes, however we would preferably also include information about currently offline nodes as we assume that they are rejoining the system at some point. This creates another problem which has to be considered. How long should information about offline nodes be kept and included in our estimates? This needs to be addressed because some nodes will in reality never return to the system or even if they return, the information the system has kept in the mean time will be stale and might no longer reflect the behaviour of the offline node.

Next we need to consider how to deal with changing node behaviour.

We can not expect a node to always follow one pattern, and hence a node's mean *Ton* and *Toff* time will change over time. In [15] and [25] the authors propose to solve this problem by periodically rerunning the protocol (preferably in a concurrent manner as to always have up to date estimates), this however adds a lot of extra complexity especially in the presence of churn. It also requires some sort of synchronisation mechanism which also adds complexity. Hence we would preferably device a system which can deal with changing node values in a single instance of the protocol, and still be accurate.

Lastly we need to consider how to do estimation and aggregation over the privacy preserving links without compromising node privacy, and undermine the strength of the pseudonyms, meaning that an attacker should not be able to track individual nodes across instances of pseudonyms by the values it is transmitting.

Summing up, our protocol for estimating churn should have the following properties:

- A way for nodes to choose a data set for the estimates which includes values from offline nodes
- A way of limiting the time values from offline nodes are kept in the set
- An efficient way of aggregating data values for the use in estimation
- A way of updating aggregation data values without having to restart the protocol
- A way of aggregating data for estimation without compromising node privacy

6.2 Assumptions

We assume that nodes have the ability to observe their own behaviour pattern, and can calculate its own mean T_{off} time. We have implemented this capability by having each node store the last x T_{off} times it has observed about itself, and then calculating the mean of these x observations. This scheme is quite simple and more complex algorithms could be applied if necessary but for this work we found this to be sufficient.

We assume that all nodes are behaving according to the protocol and that no Byzantine behaviour is present, maliciously or otherwise.

The solution is set within the application context described in Chapter 4 and we assume the presence of every aspect described in [33], especially regarding pseudonyms, overlay creation and overlay maintenance. A brief description of these are given in Section 4.3.

The only difference from the assumptions made in [33] is the prior knowledge of mean *Toff* time in the network, as we are trying to estimate this.

6.3 Solution

Table 6.1: Definitions used for describing the algorithm

Parameter	Description
O	A node's observation about its own *Toff* time
TN	A node's trusted peers
PseudoEstimateSet	A node's selected set of pseudonyms which has a configurable size
PseudoCache	A node's cache of pseudonyms for gossiping which has a configurable size
PseudoGossip	A set of pseudonyms sent when gossiping which has a configurable size
P	A node's pseudonym
T	The lifetime of a pseudonym
E	A node's estimate of the mean *Toff* in the system
C	A logical counter for an estimate indicating how new its age
i	A uniformly random number.

Table 6.2: Properties of a node

Parameter	Description
O	Its observation about its own *Toff* time
TN	The trusted peers
PseudoEstimateSet	The set of pseudonyms used for estimation
PseudoCache	The cache of pseudonyms for gossiping
P	The pseudonym

Table 6.3: Properties of a pseudonym

Parameter	Description
T	The lifetime of a pseudonym
i	The pseudonym's ID
{E, C} pair	*E* is an estimate made by the holder of P
	C is a logical counter indicating when E was made

Our solution for the problem described in Chapter 5 is reusing many of the elements found in the protocol described by [33]. We first describe our solution using the abbreviations presented in Table 6.1, before presenting the reasoning behind our choices in Section 6.4.

The pseudonyms have the same properties as the pseudonyms described in Section 4.3.1, and the PseudoCache is equal to *pseudonym cache* described in Section 4.3.2. The gossiping algorithm is also the same. The PseudoEstimateSet described in Table 6.1 is equal to the *pseudonym links* described in Section 4.3 and the sample selection is done in the same manner as described in Section 4.3.3. This means that the same pseudonyms are being gossiped and the same pseudonym links are selected as in [33]. Essentially the network overlay is the same as before, and the estimation protocol is as robust as the overlay as they are for all practical purposes the

same. What is new are the addition of estimates, self observations and logical counters indicating estimation time. It is also new that the pseudonym lifetimes are determined by the estimation in stead of prior knowledge of churn. Table 6.2 defines the data structures of a node, and Table 6.3 defines the properties which are attached to a pseudonym.

6.3.1 Initial state of a node

- PseudoEstimateSet and PseudoCache will not contain any pseudonyms
- O will hold a default value
- E will be equal to O
- The node will create a pseudonym (P) for itself
- The lifetime (T) of that pseudonym will be thee times E

6.3.2 Updating *O*

Whenever a node becomes online after a period of absence it will recalculate its own O after the principles described in Section 6.2.

6.3.3 Calculating *E*

E is calculated as the mean of O and the Es which are attached to the Ps in the PseudoEstimateSet. This is calculated as E = (O + SUM(E in PseudoEstimateSet) / (NUM(E in PseudoEstimateSet) + 1)).

6.3.4 Dealing with expired pseudonyms

Before every gossiping-cycle, or before making any calculations, a node will check the T attached to every P in its PseudoEstimateSet and PseudoCache and remove any Ps which have expired. If its own P has expired it will generate a new P for itself, giving it the lifetime of thee times the current mean of the E's in the PseudoEstimateSet and resetting the counter C to 0. If PseudoEstimateSet happens to be empty at the time of creation it give P a lifetime of three times O.

6.3.5 Gossiping

Whenever a node *a* initiates a gossiping sequence it will choose a random node *b* from either its PseudoCache or TN. The two nodes will then go through the following steps:

1. Update their own estimate
2. Select a PseudoGossip set to send
3. Exchange PseudoGossip sets

4. Determine which estimates are the newest
5. Apply PseudoGossip set to the PseudoCache
6. Apply PseudoGossip set to the PseudoEstimateSet

Updating their own estimates

Whenever a node is about to gossip its own P, which it does whenever it gossips, it will check if its E has changed since last time it gossiped. If it has changed it will replace the {E, C} pair attached to its P with a new {E, C} pair where E equals the new E, and C is incremented by 1 from the previous C.

Selecting a PseudoGossip set

Nodes will exchange up the number of P slots in the PseudoGossip set. A PseudoGossip set always includes the sending node's P and will never include duplicate Ps. The remaining slots are filled by randomly selecting Ps from node's PseudoCache until either all Ps in the PseudoCache have been included, or until there are no more empty slots in the PseudoGossip set. This is the same technique as presented in [35] and used in [33].

Determine which estimates are the newest

Upon receiving a PseudoGossip set a node will first search its PseudoCache and PseudoEstimateSet for any duplicated Ps. For every duplicated P it finds, it compares the C in the {E, C} pair attached to the Ps with each other. It then replaces the {E, C} pair which contains the lowest C with the pair containing the higher C. In case of a tie it does nothing. This ensures that only the newest estimates are kept.

Applying a PseudoGossip set to the PseudoCache

Next it compares the Ps in received PseudoGossip set with the Ps in its PseudoCache and removes any duplicates from the PseudoGossip set, as well as its own P if it is present. If its PseudoCache contains any empty slots it will then fill the empty slots with Ps from received PseudoGossip set until the PseudoCache is full. If PseudoCache is, or becomes full, Ps from PseudoGossip set will replace the Ps in the PseudoCache which where chosen to go into the PseudoGossip which set which was sent during this gossiping sequence. These are the same principle for gossiping described in [35] and used in [33].

Applying a PseudoGossip set to the PseduoSet

For every P in remaining in the PseudoGossip set, the a node will compare the id (i) of P with the id (i) of the P currently holding a first slot in the PseudoEstimateSet. If the slot is currently empty the P will be inserted into that slot. If there already is a P holding the slot the ids are compared

to the i which is attached to the slot, and the P which has the id which is numerically closest to the i of the slots gets the slot. In case of a tie the P with the longest remaining T is chosen. This prosecess is then repeated for every slot in the PseudoEstimateSet. This is equal to the technique for selecting *pseudonym links* used in [33].

6.4 Reasoning behind design choices

In this section we elaborate on the choices we made to arrive upon the protocol presented in Section 6.3. Some of these choices where clearly made with the application context in mind and we elaborate some on how our solution could be used in other application contexts in Section 6.6.

6.4.1 Choosing a selection

As established in Section 6.1 we needed a way to include offline nodes into our estimates. As far as we know the only good way of doing this is to have each node have data values from a selection of nodes which includes offline nodes. This is similar to the approach taken in [2] with the exception that we are also including offline nodes. However, unlike [2] we would like to apply our technique to an unstructured P2P network. This means that in addition to including offline nodes we also need a way of choosing nodes in a such a way where every node appears about equally often in the samples. If this is not the case estimates would be skewed towards the values held by the nodes which appear more frequently than others. It is however not crucial to ensure that individual nodes do not appear multiple times within the same sample, as established in [13].

Our choice for selecting the individual samples fell on the principles outlined by [4]. Using this approach we can have nodes make a selection which statistically includes every node equally often into the selections. It has the advantage of working in any P2P network regardless of how the overlay is created, and as shown in [33] it can be used to create an overlay networks which includes offline nodes. An added benefit is that within our application context using this selection method does not add any computational or message complexity.

We have no definitive answer for the best size of the sample. As pointed out in [2] a bigger sample is better for accuracy, but can potentially result in more stale information or higher message complexity to ensure freshness in the sample. In [33] they found 50 to be a sufficient number for creating a robust network overlay, and we found this to be sufficiently big for our experiments. We do however explore the impact of different sample sizes in Section 7.2.5.

6.4.2 How long to keep values from offline nodes

Determining how long a node's value should remain in a sample is also a non trivial task. Having sample values being valid for a long time is good

for capturing values from nodes which are offline for long periods of time. The drawback of this is that some nodes will never rejoin the system and thus no longer relevant information is kept for longer. Time is also a relative concept.

Following these arguments, how long estimates should be kept in a sample should be relative to the estimated churn in the network. In [33] they showed that having pseudonyms for the overlay links lasting three times longer than the mean *Toff* time in the network was sufficient for connectivity. This seemed like a reasonable time to keep the estimates, though we also explore other times in Section 7.2.4. Returning to our application context using the same ratio for pseudonym lifetimes as how long to keep values in our estimate set allows us to couple these times saying that a value for a certain pseudonym (each pseudonym represents a node) can be kept as long as the pseudonym is valid.

6.4.3 Disseminating and updating values

Disseminating the values for the observation sets requires a gossiping technique which allows for gossiping values from more than one node at the time. If we where to limit nodes to only gossiping information about themselves dissemination would be painstakingly slow, likewise gossiping information about all nodes is of course also not practical if the network is big. Thus we need a gossiping technique which gossips a set of data in an efficient and robust manner. Our choice landed on the protocol proposed in [35]. It is light weight, suits our needs and has the added benefit of being the gossiping technique used in [33].

Having found a suitable gossiping technique we needed to address how to update the values in the estimate sets. The easiest way would be to simply have every node attach information about its *Toff* time every time it creates a pseudonym for itself and not update this value until the next time the node creates a pseudonym. This would however result in estimates being made with increasingly stale information (assuming that pseudonyms last for a while). In stead we let each pseudonym be coupled with a {value, counter} pair. The value would as the name implies represent the churn value while the counter would represent a logical counter indicating when the {value, counter} pair was created. This allows a node to update the value whenever it is gossiping its pseudonym, indicating a new value by increasing the logical counter from the previous iteration.

Using this formula aggregation data can be disseminated in the same way as proposed in [33]. Using the same formula the aggregation receives the same robustness to churn as the network overlay gets.

6.4.4 Estimation

Having established techniques for selecting a sample set and disseminating and updating data through out the network we turned our attention to the actual estimation. We could now use the technique proposed in [2] as

we have a sample created in an similar fashion. The problem with using this solution would be that every node would have to disclose information about itself in a non privacy preserving manner. This would make it easy to track individual nodes across pseudonyms, undermining the privacy-preserving capabilities of the system. Our solution to this problem is to borrow the averaging technique from [15]. In stead of every node gossiping its own churn value it will in stead calculate a mean between its own value and all the values which it has in its sample set. This is done in such a way that the self observation will count for $1/(N + 1)$ part of the mean, where N is the number of nodes in the sample set.

The benefits of doing this are twofold. First, we are addressing the variation in estimates made by individual nodes which is present in [2]. This is because all values will converge towards the same number which is one of the strong-suits of [15]. The estimates will not be perfectly even as they have to include the nodes self observations, but a lot more uniform than the distribution in estimates found in [2]. Secondly, by having every node gossip approximately the same value over time tracking individual nodes across pseudonyms becomes harder. Thus we are providing the nodes with an anonymity set.

Another neat thing about our approach is that we can achieve a uniform estimate like in [15] while being able to update individual values over time and not have to restart the protocol. The ability to adapt to changes in behaviour patterns by the nodes should also help in providing strong anonymity sets as estimates will fluctuate a bit as a result of the protocol by it self.

It is worth noting that when a node estimates the mean T_{off} time in the network for the purpose of using the estimate for other means than gossiping it will not include its self observation into the estimate. This is because doing so would skew the results towards it own self observations which is not desirable. More on this in Section 7.2.3.

6.5 Added computational and message complexity

Our proposed solution does not create or transmit any additional messages compared to the protocol proposed in [33]. It does however slightly increase the size of every message. How much is dependent on representation but if done correctly it should at most double the payload of a gossip-message. This makes the size of l, ie how many pseudonyms we transmit in every shuffle, the determining factor as far as message size is concerned. This is no different than the existing protocol. There is of course also some added computational complexity especially when comparing and applying observations when gossiping, however if done in a correctly this should be no worse than **O(M log N)** where **M** is the number of pseudonyms which get gossiped and **N** is the number of pseudonyms in the cache plus the number of pseudonyms in the overlay links.

6.6 Usage outside the application context

Our approach could in theory be applied to estimate churn in any P2P network. The gossiping method is probably the most easily interchangeable part. The only requirement the protocol itself needs is some way of getting the estimates to the nodes which are including them in their estimation sets. Preferably the gossiping algorithm should also provide some means by which provide updated estimates to the sets. We chose to use the algorithm proposed by the authors of [35] as it is robust and lightweight and was already in use, but other methods could in principle also be used. We would recommend considering using the same gossiping and sampling techniques which are used for creating the network overlay as these mechanisms typically are robust as they have to deal with churn.

The mechanism for selecting an estimate sample is a bit more critical. The samples should be selected in such a way that every node appears roughly an equal amount of times when combining all samples. In most structured network this could simply be the network overlay combined with some method for including offline nodes. Or if one does not wish to include offline nodes one could maybe improve on the idea presented in [2] with our averaging technique.

The benefit of not including offline nodes into the estimates set is that one does not have to deal with how long estimates from offline nodes should be kept in the sets. The drawback is that one is making estimates based on less potentially relevant information (this is dependant on your churn model).

If one does choose to include nodes which are not currently online into the estimation sets one has to have some common notion of time and an algorithm which guarantees that estimates from nodes which have not been online for the set time period really do get removed from the system. In our protocol this guarantee is made by the lifetime of the pseudonyms, but other approaches are definitively possible. If the nodes have a common notion of time one could for instance attach a timestamp in stead of the logical counter to each estimate, and then remove any estimate which has reached some time limit.

6.7 Summary

In this chapter we have outlined our solution for estimating mean *Toff* in a privacy-preserving manner. We propose that every node can estimate the mean *Toff* in the network by aggregating a sample of estimates from other nodes. These samples can include estimates from nodes which are currently offline, while having mechanisms in place to ensure that estimates from nodes which have been unavailable for sufficiently long are disregarded. We have also proposed a mechanism for updating estimates from individual nodes over time to reflect any changes churn patterns. Our solution is closely intertwined with the protocol proposed in [33], but as discussed in Section 6.6 the techniques could also be applied to other P2P

networks as long as have mechanisms for meeting certain requirements.

Chapter 7

Evaluation

In this chapter we first present the configuration parameters for the experiments and then define the performance metrics for connectivity and estimation, before presenting evaluations of the protocol under different conditions. This is followed by a brief summary before we present our performance metrics for evaluating the privacy-preserving aspects of our solution. We then go on to evaluate our solution by these performance metrics, rounding up the chapter with a brief summary of the performance of our solution.

7.1 Experimental environment

For our experiments we assume the presence of a perfectly working anonymity and pseudonym service which allow for creation of high bandwidth low latency links between nodes. Hence messages between nodes are assumed to happen in a short time as long as both ends of a link are online.

The time unit for our experiments is the frequency of how often the gossiping protocol gets initiated by a single node, called a *shuffle period*. This is chosen as the time unit because all important interactions in our protocol happen at this frequency.

All experiments where run on our custom simulator, which is an event driven simulator, where an event can happen at any time within the duration of a single shuffle period. Shuffle periods are not synced between individual nodes, but all nodes will shuffle at the same frequency as long as they are online.

7.1.1 Trust graph

To simulate the social bonds between nodes we create a *trust graph*. The graph is obtained by using the work presented by [36] which in turn was created crawling friend lists of Facebook users. This gives us a good real-life basis for the trust relations between nodes. From the Facebook trace we in turn started at a random node and crawled the graph selecting a number of contacts from each node until we met our desired number of nodes.

The edges in the trust graph are the same as the edges in the Facebook graph among the selected nodes. Meaning that if node a and b are friends in the Facebook graph and both nodes are selected in our sample they will be considered to be trusted by each other in our trust graph. It is also worth noting that as our trust graph is created traversing existing nodes starting from a single point, we have a connected network as the trust graph is a undirected graph.

7.1.2 Churn

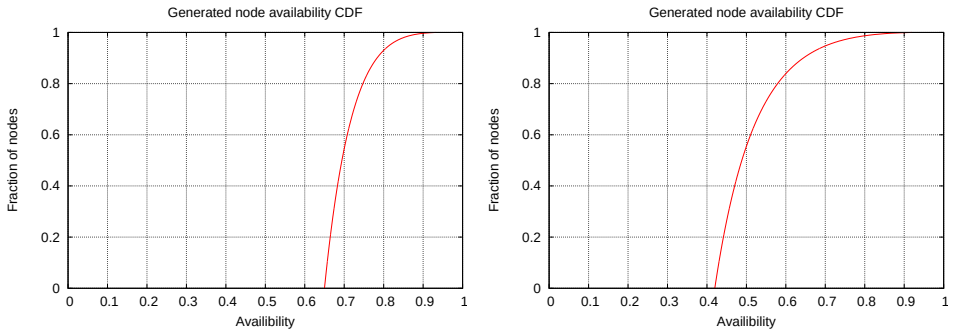
We assume that nodes have strong incentives for returning to the network because they have common interests. Hence we model churn as the time a node is online and how long it is offline. A node's online time is how long it will remain online once it has become online, hereafter T_{on} and its offline time T_{off} is how long it remains offline once offline.

Artificial churn

Every individual node has a mean T_{on} and T_{off} time, but individual session times will wary with an exponential distribution as proposed in [38]. A nodes availability is calculated as $Availability = T_{on} \ / \ (T_{on} + T_{off})$ where T_{on} is mean online time and T_{off} mean offline time. As we are trying to estimate the T_{off} time we have given nodes a fixed T_{on} time unless otherwise specified for all experiments presented in this thesis. The CDF of the availability for the different experiments can be seen in Figure 7.1. No nodes have more than about 90% availability and few or none have less than 10% availability.

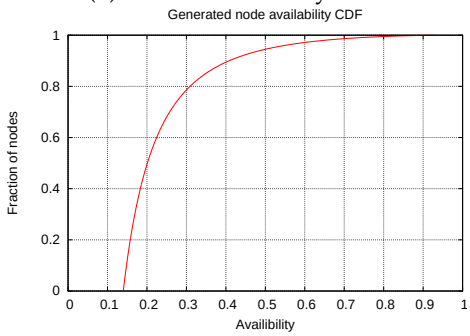
Real-life churn traces

To show that our protocol can work in a real life scenario we ran the experiment using real-life churn traces from Skype presented in [31]. A closer look at the traces revealed that a significant amount of nodes had less than 1% availability as can be seen in Figure 7.2a. We felt that running the system with so many nodes which had such a low availability would not show the true potential of the system. Thus we removed the nodes with availability less than 2% giving us the availability distribution shown in Figure 7.2b. Doing so also meant that the mean availability of the nodes went from 33.5% to 43% availability. To further adapt the trace to our notion of time which is shuffle periods, we converted the values to be equivalent of having a shuffle period every minute. This is because the trace has relatively long session times and our protocol requires multiple sessions to get an accurate number for the self-monitoring. We further replicated the session times by repeating the churn patterns multiple times to get sufficient number of shuffle periods as one run would only be equivalent to proximately 3900 shuffle periods.

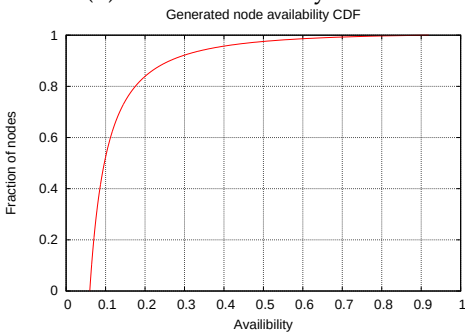


(a) Mean availability 70%

(b) Mean availability 50%

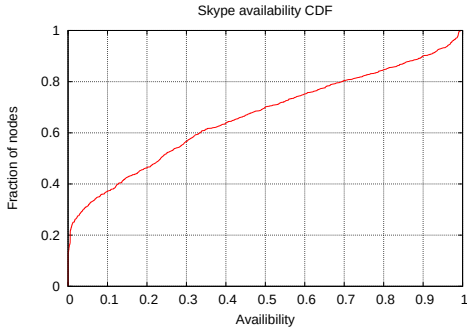


(c) Mean availability 20%

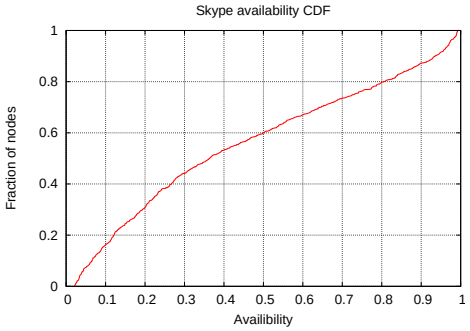


(d) Mean availability 10%

Figure 7.1: CDF distribution of availability for nodes used in experiments



(a) Before removing nodes



(b) After removing nodes

Figure 7.2: CDF distribution of node availability in Skype traces from [31]

7.1.3 Default configuration parameters for proposed solution

For the majority of our experiments we have a set of default configuration parameters, summarized in Table 7.1. These parameters can be assumed as the settings for the experiments shown in this thesis unless specifically stated otherwise. We have also included a range column for the values which we do vary in some experiments.

As in [33] we have chosen a graph size of 1000 nodes. In [33] they tried other sizes for the trust graph, but reported to find similar results with the 1000 node graph as with other sizes, and hence we have found 1000 nodes to be a suitable number of nodes. The mean duration of T_{on} time has been

Table 7.1: Default values for parameters in proposed solution

Parameter	Default	Range
Nodes in trust graph	1000	
Mean T_{on} time in shuffle periods	10	
Pseudonym lifetime ratio (* T_{off})	3	3 - 9
Pseudonym cache size	400	
Number of pseudonyms exchanged during a shuffle	40	
Target number of pseudonyms in overlay links	50	25 - 75
Number of backups pr pseudonym in overlay links	5	

set to the equivalent of 10 shuffle periods for all experiments not using real-life churn traces. Increasing this number does not significantly increase or change the performance of the system, however reducing this number will at some point effect performance. We have not explored where this limit goes as we consider this outside the scope of this thesis.

The pseudonym lifetime has been set to three times the estimated mean T_{off} time at the time of pseudonym creation. This means that pseudonym lifetimes will change over the duration of an experiment. Three times mean T_{off} was chosen as a good number based on the work presented in [33].

With regards to shuffling we have a shuffle cache size of 400, and exchange up to 40 pseudonyms in every gossip exchange. Every node will try to get 50 pseudonym links in their overlay links, in addition to their overlay links from the trust graph. Finally, every node tries to have up to five backup pseudonyms for every possible pseudonym in the overlay links. This helps with the connectivity at the beginning of the system and with nodes that are offline for long periods at the time, but has no notable effect on the majority of the nodes once the system has stabilized.

Nodes are also given the time equivalent to 100 shuffle periods to monitor themselves before we start gossiping. Self monitoring is done by monitoring and storing the last 50 T_{off} times in a ring-buffer and then calculating the mean of these values. Initially nodes are given a value equivalent to 90% availability, however this value gets overwritten by the first value gained from self-monitoring.

7.1.4 Performance metrics for estimation

To show that our solution works we have two main goals. Firstly, we want to show that our system can deliver a comparable performance to the systems presented in [33], despite the nodes having no prior knowledge to the mean *Toff* value and having to estimate this. Estimation can never be as good as working with perfect information, which was the basis for the experiments presented in [33], but good estimates can come close.

To to this we make comparisons in terms of node connectivity and mean path length. Connectivity is calculated by checking how many of the currently online nodes are connected to each other in terms of connected components. If node A and B are both online and there is a way for node A

to communicate with node B either directly or through intermediate nodes, they belong to the same component in the graph. All nodes which can communicate with either (and hence both) are also connected to the same component. We then define the connectivity as the proportion of online nodes which are connected to the largest connected component. Mean path length is calculated by doing a BFS from all nodes in the largest connected component.

The second goal is that every node should accurately estimate the same T_{off} value, regardless of its own T_{off} value, and that this value should be correct. It is also desirable that this should be achieved in a relatively short timespan. To show this we provide two main metrics for evaluation. Firstly how close is the mean of the estimated values to the mean of the values gotten from self-monitoring over time. This is done by checking at certain time intervals which value every single node would estimate as the system wide mean T_{off} time and at the same point in time ask what its own T_{off} time derived from self monitoring is. When averaged these numbers show us how close our estimate is to the actual mean and how this develops over time. This in turn allows us to see how fast the algorithm is converging.

We define convergence to be achieved either when the estimated values are no longer changing, or the fluctuations in the estimates are showing a recognisable pattern.

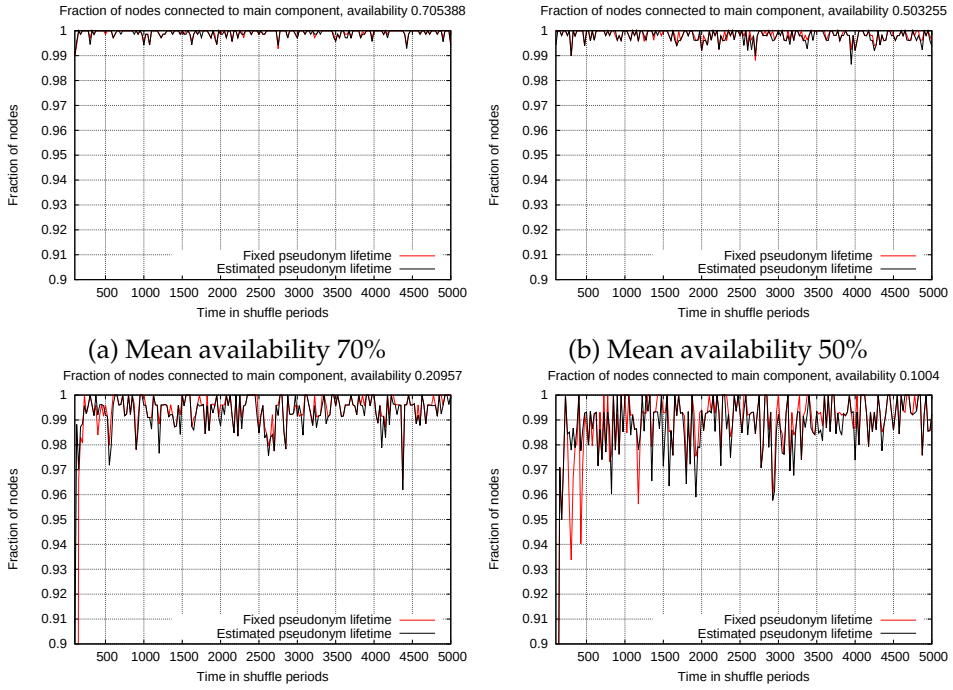
The second metric looks at the same values at certain points in time and look at the distribution of the estimates compared to what the self-monitoring shows. Coupled with the first metric this allows us to show both how accurate the estimates are and that the variance in the estimate distribution is low which indicates that all nodes are estimating well. We also have some additional performance metric for privacy-preservation which are covered in Section 7.4.

7.2 Experiments

In this section we are first showing that the system presented in [33] still works, even when we are estimating the amount of churn in the network. This is done by comparing the connectivity of the systems, presented in Section 7.2.1, and comparing mean path lengths in the network, presented in Section 7.2.2. We are then showing how our proposed protocol for estimation is preforming under different scenarios. The main metrics here are speed of convergence, accuracy of estimates and distribution of estimates. We first present the performance using the default parameters presented in Table 7.1, which are presented in Section 7.2.3. We then take a closer look on the impact of pseudonym lifetime on the accuracy, convergence and distribution in Section 7.2.4 followed by the impact of the number of overlay links in Section 7.2.5. Finally we show how the system is performing when run with churn traces taken from a real life system.

7.2.1 Connectivity

To compare the connectivity of our protocol to the work presented in [33] we ran the same experiments in two different scenarios. In the first one, we gave every single node prior knowledge of the actual mean *Toff* time across all nodes. This means that pseudonym lifetimes generated by all nodes is always the same for all nodes. This is equal to the conditions found in the experiments presented by [33]. In the second scenario none of the nodes had any prior knowledge and we used our proposed protocol to estimate the mean *Toff* time in the system based on the numbers that the nodes got from self monitoring. Otherwise the two sets of experiments where exactly the same.



(a) Mean availability 70%

(b) Mean availability 50%

(c) Mean availability 20%

(d) Mean availability 10%

Figure 7.3: Node connectivity over time

As we can see from Figure 7.3 the connectivity of the nodes in the system is similar in both scenarios with slightly grater deviation for the lower availability. This becomes more apparent when we look at the mean connectivity of the systems over the duration of the experiment as seen in Figure 7.4. The slight decrease in connectivity that we are seeing from the proposed algorithm stems from the algorithm slightly underestimating the mean *Toff* time, and as a result the pseudonyms which get generated do not last as long as they would when using the fixed value. This is discussed further in Section 7.2.3.

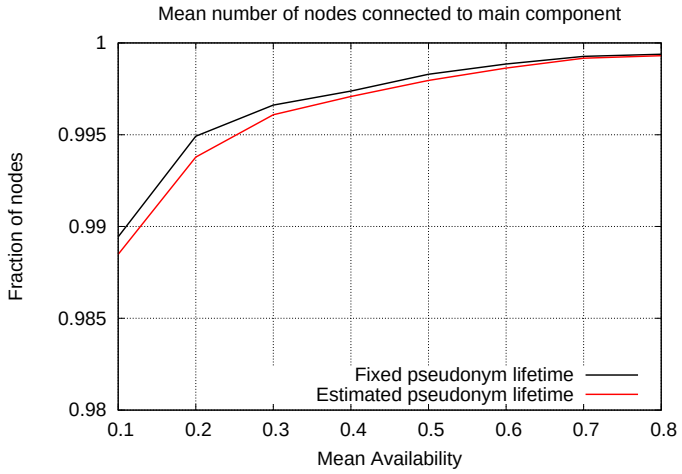


Figure 7.4: Mean connectivity for the duration of the experiments

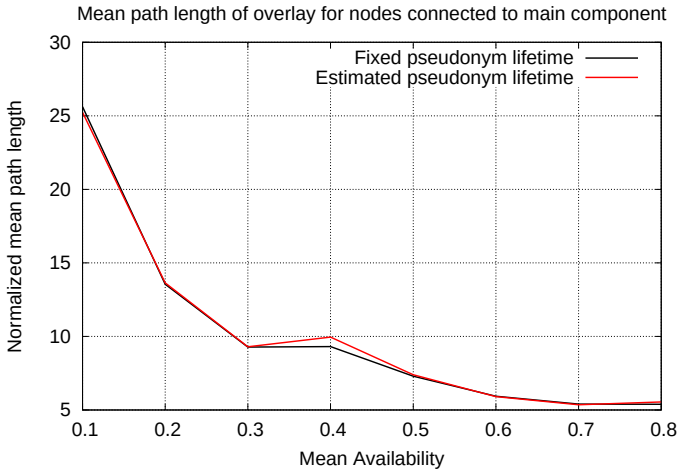


Figure 7.5: Normalized mean path length for the duration of the experiments

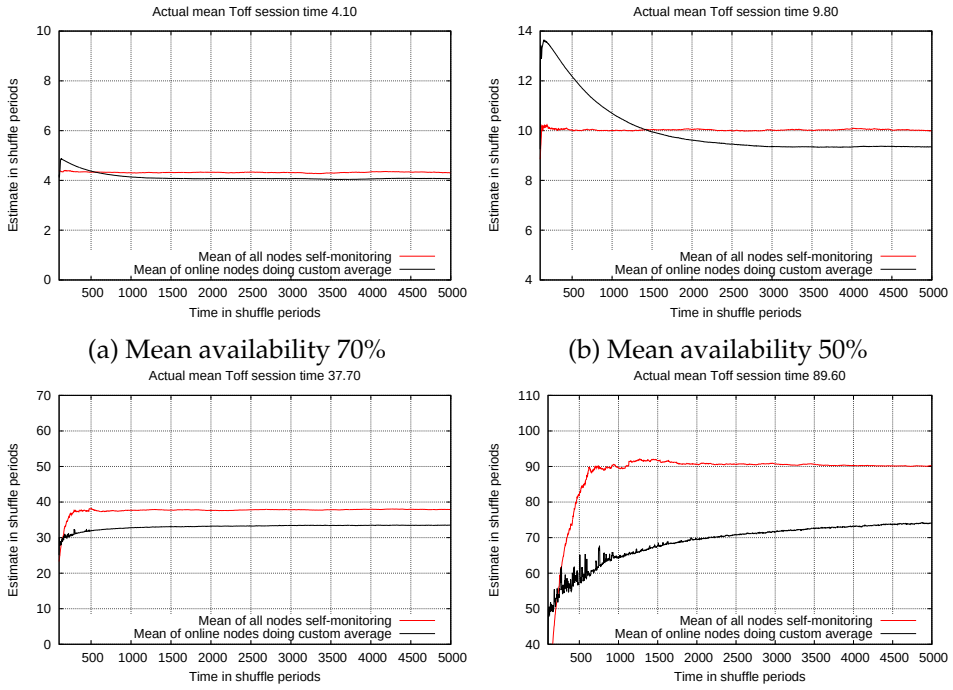
7.2.2 Mean path length

As we can see from Figure 7.5 the normalized mean path length of the nodes in the single biggest connected component of the system also remains almost unchanged when estimating mean *Toff* compared to having a fixed *Toff* value. Combined with the results shown in 7.2.1 we see that the performance of the system is essentially unchanged from introducing estimating *Toff* compared to nodes having prior knowledge of these values.

7.2.3 Performance under artificial churn while self monitoring

In this section we focus on the performance of the proposed algorithm in terms of our second group of performance metrics, namely that the nodes are estimating the same *Toff* value and that this value is correct. This is done using the artificial churn presented in Section 7.1.2.

As we can see from Figure 7.6 the algorithm converges fairly fast



(a) Mean availability 70%

(b) Mean availability 50%

(c) Mean availability 20%

(d) Mean availability 10%

Figure 7.6: Convergence of proposed algorithm while self-monitoring

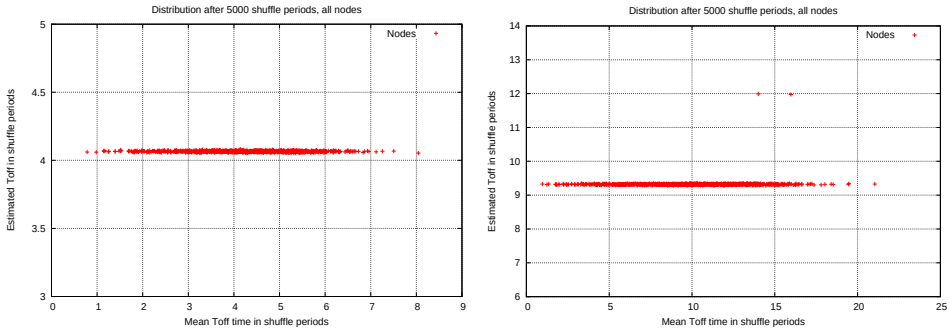
on an estimated *Toff* which is fairly accurate to the actual mean *Toff* as calculated by self monitoring. The estimates get increasingly less accurate as the availability decreases. This is because the nodes which have a low availability and hence a high *Toff* value are not online sufficiently often to get included into the mean of other nodes. Say that a node has an mean *Toff* time of 100 shuffle periods. If the mean *Toff* across all nodes is estimated to be 10 shuffle periods a pseudonym generated by our node will last for 30 shuffle periods. For those 30 shuffle periods the *Toff* value of the individual node will be present in the system, but for the on average remaining 70 shuffle periods this value will not be propagated through the network. But even with this deficiency the algorithm is still fairly accurate, even with a mean availability of 10% and a fairly heavy skew in individual node *Toff* time as visualized in Figure 7.1d.

An interesting observation can be made in Figure 7.6b. Here the estimates are to high for the first 1500 cycles or so, before stabilising slightly under the actual mean *Toff* after about 2500 cycles. Situations like these can happen if nodes have extreme values which get propagated for a long time unchanged. Consider the following scenario. A node has joined the system for the first time after a long period of absence. The node has no valid pseudonyms and has hence only the information from it's self observation to make decisions about. Based on this it creates a pseudonym with a very long pseudonym lifetime for itself. It then gossips this pseudonym and its estimate, which at this point will only be about itself, to one of its trusted links and immediately goes offline. The pseudonym will be kept in the

system for a long time as it has a long lifetime and the attached estimate value will hence be kept equally long. Normally this would not be that big of a problem as a online node would be able to propagate new estimates to the network gradually sending less extreme estimates. But a offline node can obviously not do so.

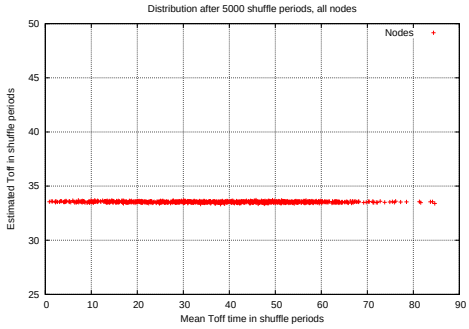
This extreme scenario can potentially happen at any point in time, but is most likely to occur before the gossiping algorithm has run for very long. If the system has been running for some time, all online nodes will most likely have more pseudonyms in their cache than they gossip during a single gossiping instance. A node with an extreme value will then in a single turn have multiple less extreme values to base its estimate on, meaning that starting from the next round of gossiping it will gossip a much less extreme estimate value. If on the other hand the nodes are in the start of the systems existence every gossiping cycle may only contain a few pseudonyms. Having less pseudonyms and hence estimate values to make an estimate from will most likely result in a more extreme value being transmitted for longer, which in turn increases the chance that the node will leave the system before proper estimate can be made.

To deal with these potential situations one should probably consider making some minor adjustments to the protocol. For instance not gossiping estimates before one has at least a certain number of other estimate values as data points could be a possible solution. It is however important that one also considers the impact any changes can make on the privacy preserving properties of the protocol if one makes changes.

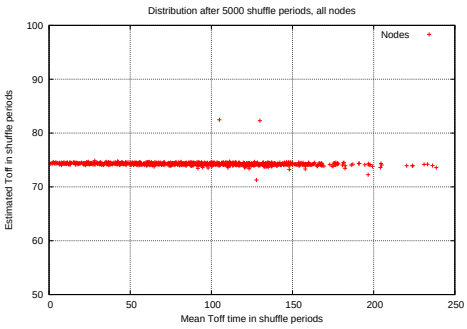


(a) Mean availability 70%

(b) Mean availability 50%

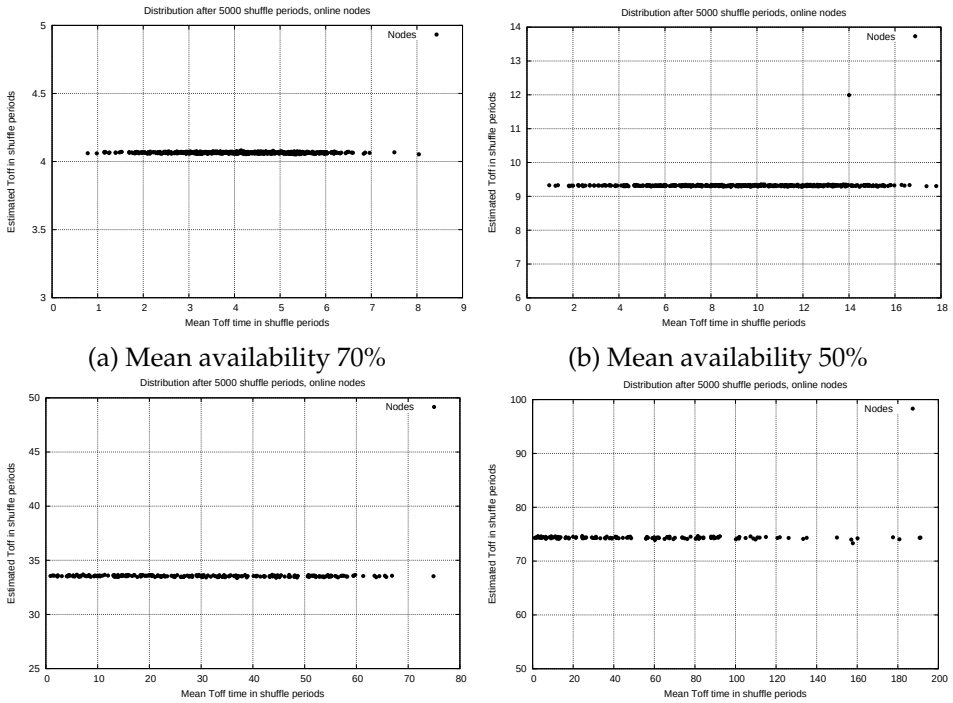


(c) Mean availability 20%



(d) Mean availability 10%

Figure 7.7: Distribution of estimate over own value for all nodes



(a) Mean availability 70%

(b) Mean availability 50%

(c) Mean availability 20%

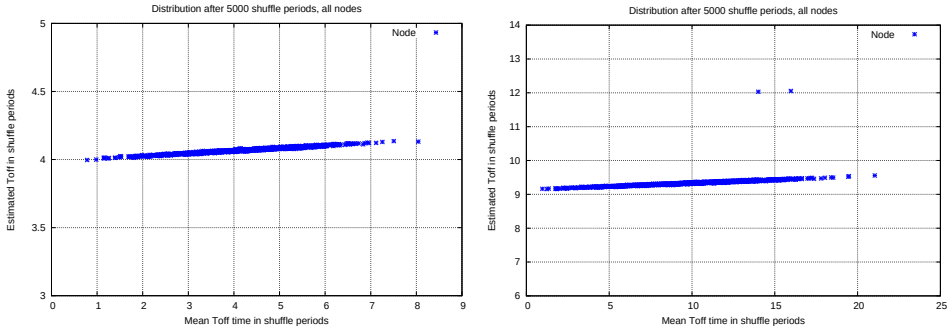
(d) Mean availability 10%

Figure 7.8: Distribution of estimated means over own value for online nodes

Table 7.2: Standard deviation of estimates among online nodes in proposed algorithm

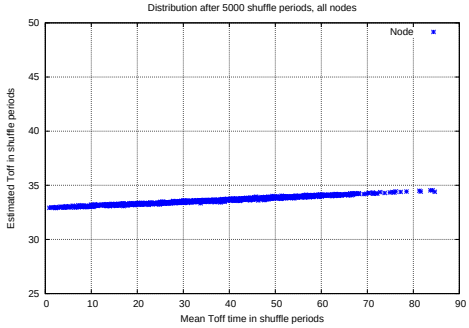
Availability	**Proposed algorithm**	**Actual mean T_{off}**
70 %	0.004	4.1
50 %	0.010	9.9
20 %	0.048	37.7
10 %	0.215	89.6

Taking a look at our second performance metric again, the distribution of the estimates, things look a lot better. As Shown in Figure 7.7, which is a snapshot of all nodes at a single point in time, we observe that the node's observations are close to uniform across all nodes. Also worth noting are the artefacts that can be observed in Figures 7.7b and 7.7d. These are nodes which are offline and have been offline for so long that most of their pseudonym links have expired, and hence have few or no other observations for making a good estimate, or they have just recently again become online after such a period, and has not yet shuffled sufficiently many times to replace the dead links. If we take the same snapshot, this time only including nodes which are online at the time of the snapshot we get the plot in Figure 7.8. As we can see the artefacts observed earlier are gone, and the estimates are even more uniform. This is observed even better in Table 7.2 which shows the standard deviation of the estimates.

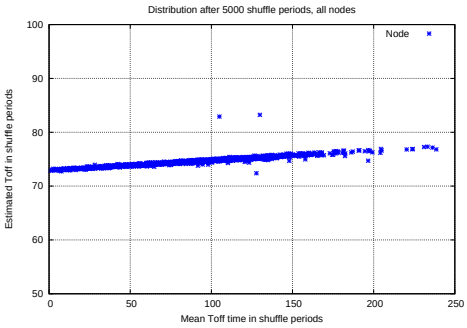


(a) Mean availability 70%

(b) Mean availability 50%



(c) Mean availability 20%



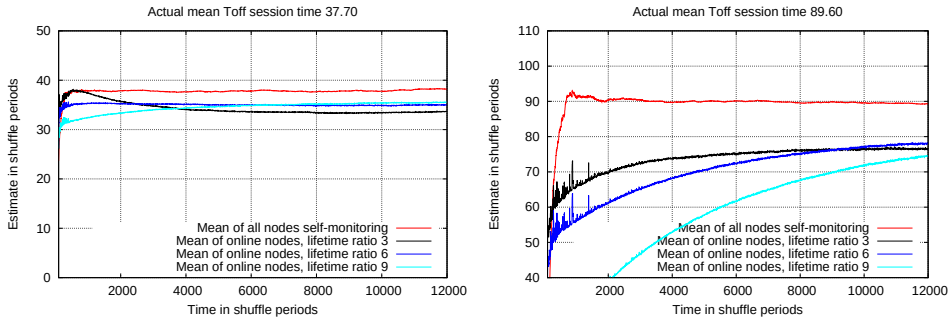
(d) Mean availability 10%

Figure 7.9: Distribution of estimated means over own value for all nodes when including own estimate in the average

As mentioned in Section 6.4.4, we do not include a node's own observation about itself when estimating what the system wide mean is. The reason for this becomes apparent when comparing the graphs from Figure 7.9 and 7.7. When including its own observation from self-monitoring we get a less uniform estimate across the nodes with a slight tendency towards a nodes own T_{off} value. This can be seen in Figure 7.9. This tendency is not surprising considering that the calculated mean includes a node's own T_{off} value, where this value in this case will make out 1/51 of the mean. What makes Figure 7.9 really interesting though is that these are the values which the individual nodes would report as its observations about itself to other nodes. This also means that the number of pseudonym links individual nodes keep in their overlay links has an effect on how accurate the estimates are. We go more into this in Section 7.2.5.

7.2.4 Pseudonym lifetimes effect on accuracy

In this section we show how the solution performs if we increase the pseudonym lifetime. We will focus on the lower availability graphs, and show how the system preforms when letting pseudonyms last 3 (default), 6 and 9 times longer than the estimated mean T_{off} time. The first thing we notice in Figure 7.10 is that the system takes a lot longer to converge the longer the pseudonym lifetimes are. The main reason for this is how the



(a) Mean availability 20%

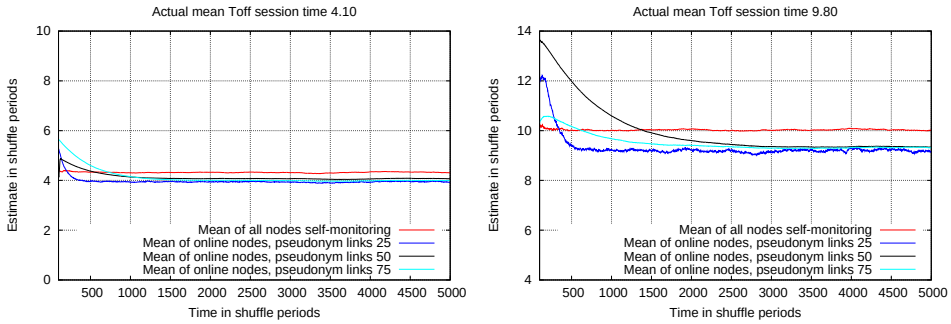
(b) Mean availability 10%

Figure 7.10: Convergence of proposed algorithm while self-monitoring, with different pseudonym lifetime ratios

overlay works. A node will only replace a overlay link once it either has a better numerical match for the slot in the overlay, or that the existing pseudonym link expires. As pseudonym lifetimes increases the time between introduction of new pseudonym also increases resulting in little change in the composition of the overlay topology. Random graphs are known for their theoretically short path lengths between nodes, but under heavy churn a lot of these paths will be unavailable for communication for long time periods creating longer path lengths between the online nodes which again results in slower convergence. In addition to this nodes will also keep stale information much longer in their overlay than they would with shorter pseudonym lifetimes as new information about estimate values also will take more time to get propagated through the network when path lengths are increasing. If one needs long pseudonym lifetimes one might want to explore the effects of additional pull mechanisms for nodes toward pseudonyms for which there has not been updated estimate values for some time. The benefit of having longer pseudonym lifetimes is that the estimate once converged becomes slightly more accurate the more we increase the pseudonym lifetime as we can see from Figure 7.10a. This is because observations from low availability nodes have more time to get disseminated through the network before they expire. And more importantly, low availability nodes are represented for longer in the overlay as their pseudonyms last longer. Nodes also do not need to be online as often to renew their pseudonyms and still maintain the same presence in the overlay. The distribution of the estimates are pretty much the same as before.

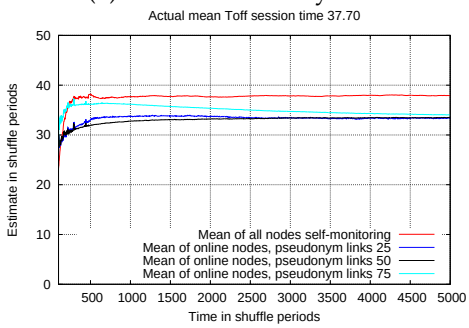
7.2.5 Overlay sizes effect on accuracy

The number of pseudonym links each node maintains will of course have a certain effect on how accurate the estimates are. Generally speaking more is better in terms of accuracy as we are basing our estimates on more information, however with a bigger overlay we also create more computational overhead for each node. In Figure 7.11 we have plotted how the system preforms under varying numbers of pseudonym links. The

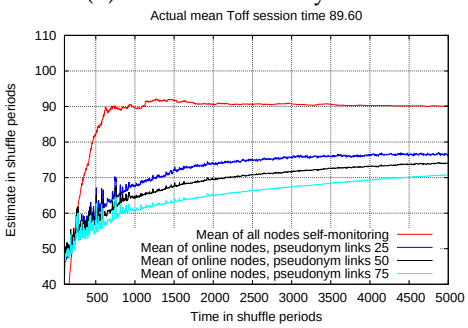


(a) Mean availability 70%

(b) Mean availability 50%



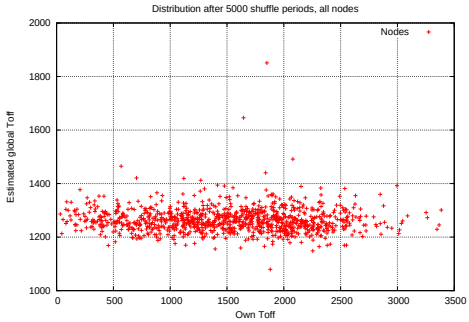
(c) Mean availability 20%



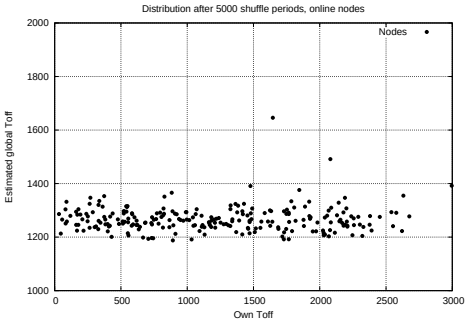
(d) Mean availability 10%

Figure 7.11: Convergence of proposed algorithm while self-monitoring T_{off}, with different number of pseudonym links

most apparent trend is that when reducing the number of pseudonym links in the overlay links to 25 the accuracy of the estimate suffers. The estimates are also less stable over time. This is because if one of the pseudonyms expire a node will replace a much bigger part of its base than with bigger number of pseudonym links.



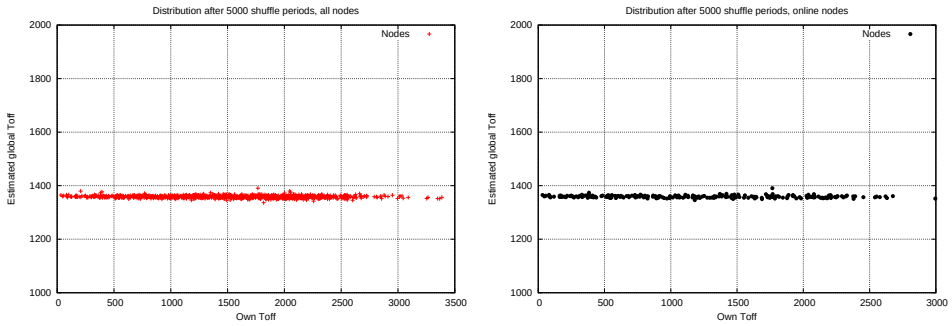
(a) All nodes



(b) Online nodes

Figure 7.12: Distribution of estimated mean over own value, 25 pseudonym links, availability 20%

And as we can see form Figure 7.12 the distributions of the estimates become less uniform when decreasing the number of pseudonym links. Looking at the performance when increasing the number of pseudonym links we see that the estimates are slightly more accurate. However, as the



(a) All nodes (b) Online nodes

Figure 7.13: Distribution of estimated mean over own value, 75 pseudonym links, availability 20%

Table 7.3: Standard deviation of estimate among online nodes with different number of pseudonym links pr node

Availability	**25 links**	**50 links**	**75 links**	**Actual mean T_{off}**
70 %	0.047	0.004	0.004	4.1
50 %	0.330	0.057	0.015	9.9
20 %	0.143	0.122	0.031	37.7
10 %	0.365	0.166	0.148	89.6

availability decreases the time it takes to converge increases significantly when we increase the number of pseudonym links. The reason for this is a bit similar to when we increase the pseudonym lifetime. We will store more stale information. This time it is not for a longer period of time, but from more nodes, and frequently online nodes will make up a larger proportion of the individual overlays as their pseudonyms will be replaced quicker when they expire, skewing the results for longer. Once the estimates have converged this is no longer a problem as low availability nodes have a sufficiently big window in which to replace a pseudonym. This of course assuming that not all pseudonyms from low availability nodes expire at the same time. The standard deviation of estimates is also being affected by the number of pseudonym links in the overlay, getting better as we increase the number of pseudonym links as we see from Table 7.3.

7.2.6 Performance under real-life churn using Skype traces

As we can se in Figure 7.14 the estimate is off by quite a bit from the actual mean T_{off} time. This is because there is no point in time where all the low availability nodes are online at the same time which we also show in the Figure 7.14. We have also introduced another metric to compare the accuracy of the estimates to, which is the mean of all nodes which have been online at some point during the last three times mean T_{off} time in the network. The idea is that if the estimates are correct, a pseudonym should last this long (3 * mean T_{off}). The algorithm can not include

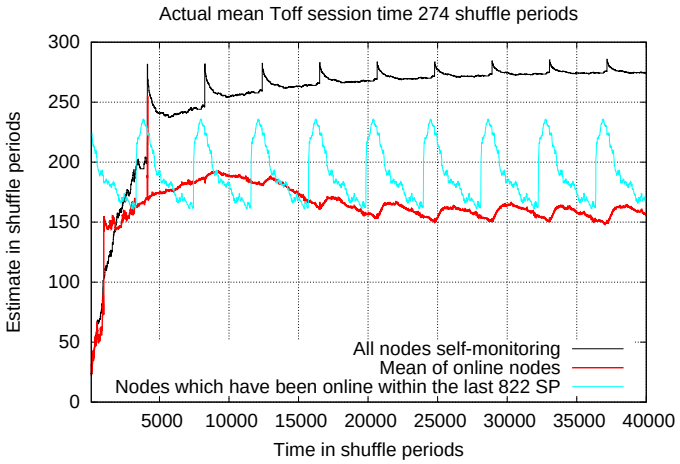
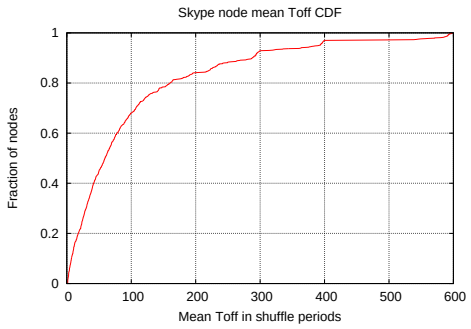
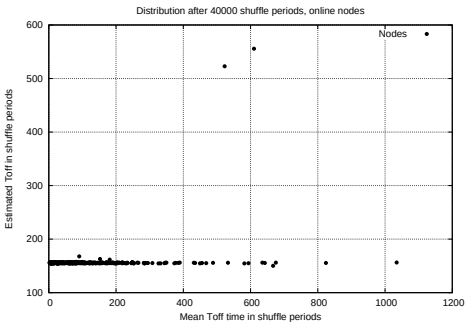


Figure 7.14: Convergence when using churn from Skype traces presented in [31]

observations from nodes which do not have a valid pseudonym in the network because nodes without a valid pseudonym can not appear in a nodes pseudonym links. To illustrate why we have to do this Figure 7.15a show the CDF distribution of the mean T_{off} times for individual nodes. If we now compare the estimate to the mean of the nodes which are actually online and the mean of the nodes which potentially should have a valid pseudonym in the network if our estimates where estimating correctly, our estimates are reasonably close. It is especially worth noting that the estimates adapts to the periodical peaks with a slight delay. This delay is to be expected as it takes a little time for new pseudonyms to get disseminated through the network.



(a) CDF of node mean T_{off} times



(b) Distribution of estimates

Figure 7.15: Mean T_{off} distribution and estimate distribution with Skype traces presented in [31]

Turning to our second performance metric again, which is the distribution of the estimates, we get Figure 7.15b. We for the first time see two artefacts, most likely because the nodes have just recently rejoined the system after a long period of absence, but otherwise the distribution is as with the previous experiments using synthetic churn.

7.3 Summary of estimation performance

So far in this chapter we have shown that it is possible to estimate the mean *Toff* time with reasonable accuracy and within a reasonable time. We have shown that we can achieve similar connectivity and path lengths while estimating as when giving the system perfect information. The accuracy of the estimates are only mildly affected by the size of the network overlay and the length of pseudonyms lifetimes, and all nodes are estimating approximately the same value regardless of their own availability with very low standard deviation. Speed of convergence is directly related to pseudonym creation and lifetime, but once stabilized the system remains stable, and as we can see from Section 7.2.6 the system will adapt to changes in the churn. We have also shown that our algorithm for estimating churn works both under quite extreme and skewed churn scenarios, as well as under churn based on real life traces.

7.4 Privacy-preservation

Estimating churn of a F2F network in a privacy-preserving manner will always be challenging. If we where to do it over the trust graph only it would take a significant time to converge, especially under heavy churn. When utilizing the overlay links we are to a certain extent compromising the privacy provided by the expiring pseudonyms, however we are converging on a correct value a lot faster, even under heavy churn. One of the neat things with our solution is that if a node does not have any valid pseudonym links it can only gossip with trusted links, where disclosing its actual mean T_{off} time is not violating privacy, and once the node has pseudonym links to gossip with, the nodes mean T_{off} gets hidden among the other observations, an anonymity set.

7.4.1 Performance metrics for privacy-preservation

As pointed out by the authors of [32] and [8] when evaluating the strength of an anonymity set one should consider both the probability an potential attacker would assign to each individual in the set as well as the number of individuals in the set. We hence need to evaluate both the strength of the set and the size of the set.

The metric for the strength of a set should be a threshold for the minimum probability an attacker would give any given individual in the set. Say that we are looking at a data set as an potential attacker, and we are looking for a individual node in this set. For each node we attach a probability between 0 and 1 for being the node we are looking for. We give node x a probability of 0.2 for being the node we are looking for. Node x would be a part of the anonymity set of the node which the attacker is looking for if the threshold is anywhere below 20% but would not be a part of this set if the threshold is higher. Having a generic way

of calculating these thresholds is important if one wants to compare the privacy-preserving capabilities of different estimation algorithms.

Over time a individual node will gossip a lot of observations to other nodes. By monitoring these values for individual nodes we can calculate the mean value transmitted by that node and then calculate the standard deviation of those observations. We define that to be within the anonymity set of each other two nodes have to transmit values within the same range. We further limit this range to any value transmitted within two times the standard deviation of two nodes means. Assuming a normal distribution this is statistically 96% of all values. By doing this we are effectively excluding any extreme values transmitted by nodes. We define the strength of the set as the probability for a message from any of the two nodes to be within this range.

Say that node A has a standard deviation of 1 with mean 1. Node B also has a standard deviation of 1 but a mean of 2. This means that according to our definition node A has a range from 3 to -1 (mean +/- standard deviation), while node B has a range from 4 to 0. The overlap between node A and node B is hence the range from 3 to 0. The probability for the nodes to transmit a value within this range is $p(x) = (Ax/A_{total}) * (Bx/B_{total})$ where x is the number of messages which fall within the overlap range and *total* is the total number of messages sent by the node.

7.5 Evaluating privacy-preservation

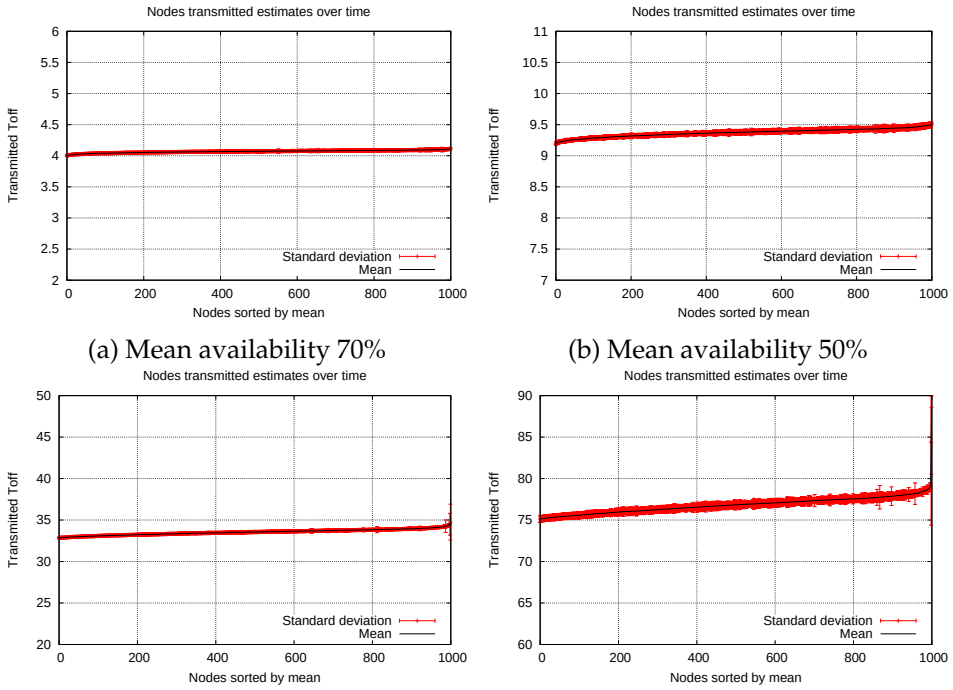
7.5.1 Set strength

As pointed out in [32] and [8] when evaluating or comparing anonymity sets the strength of the set needs to be considered. The statistically maximum set strength which can be achieved with our definition of set strength is 0.92 assuming a normal distribution of the values. This is a result of how standard deviation is calculated (See Section 2.6 for a brief introduction). We chose our definition for calculating set strength to be bound by the standard deviation to limit extreme values in the data sets to create unreasonable large overlap areas. The main drawback with our definition, as far as we can see, is that it assuming a reasonable spread of the values within the standard deviation. If nodes where to always transmit individually distinguishable values within our range then an attacker would be able to give different probabilities than the ones we are calculating he would. However, manually evaluating sets looking for patterns is not really a option when trying to compare large data sets, so we feel confidant that our metric is a good way of calculating set strengths of anonymity sets.

Achieving a set strength of 0.92 is possible if two nodes have the same mean and the same standard deviation. In this case statistically speaking 96% of the values fall within the range giving us the probability of P(x) = (96/100 * 96/100) = 0.92. To show the degree of anonymity which can potentially be provided by the algorithm we will focus on three thresholds:

75%, 50% and 25%. If we take the 50% threshold as an example. This means that for a node to be considered a part of the anonymity set of another node if you take any single value transmitted by either node there has to be an at least 50% chance that this value was within the set range. Having this guarantee we can now truly start to evaluate the anonymity provided by the number of nodes in the set.

7.5.2 Privacy-preservation under artificial churn



(a) Mean availability 70%

(b) Mean availability 50%

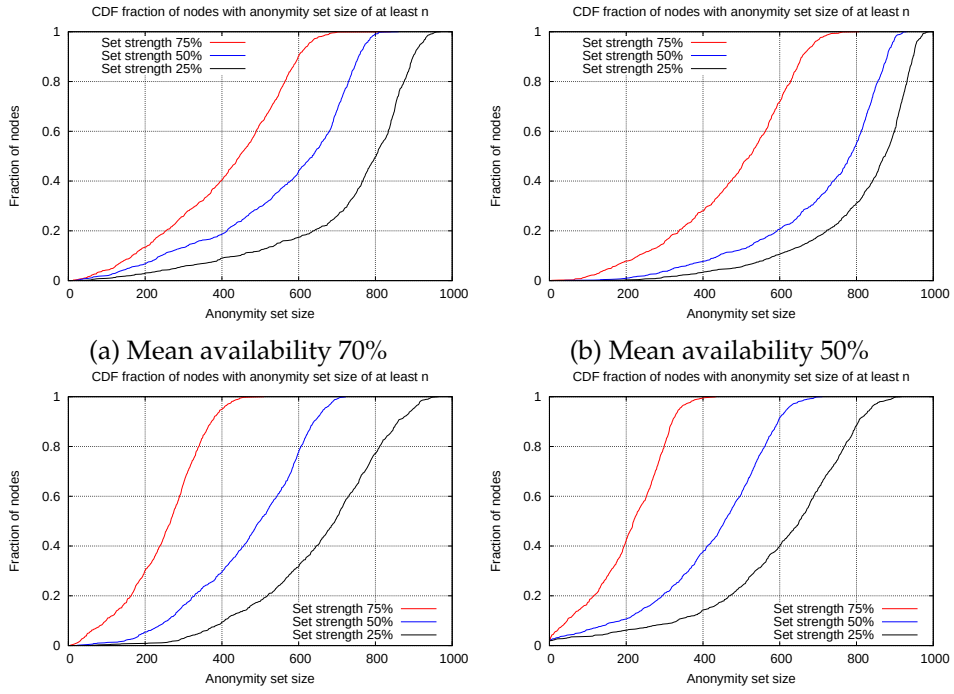
(c) Mean availability 20%

(d) Mean availability 10%

Figure 7.16: Proposed algorithm, mean estimate and standard deviation over time

In Figure 7.16 we have calculated the mean of the values which a node transmits once the system has converged with the standard deviation of this mean. As we can see from the Figure the distributions are fairly even, which indicates that tracking individual nodes across pseudonyms based by transmitted values should be fairly complicated. The exception to this are the nodes on the extreme edge of the scale. This is especially true for the low availability nodes which we find in Figure 7.16c and 7.16d.

Turning to Figure 7.17 it becomes more apparent how close the distributions are when we calculate the *anonymity set sizes* and plot them in a CDF manner. As we can see the anonymity set sizes are slightly bigger for the high availability scenarios. Focusing on the low availability scenarios in Figure 7.17c and 7.17d we see that for the 50% set strength only 10% of the nodes have a set size smaller than 200 nodes. If a 75% set strength would be required only 20% of the nodes would have a set size of less than



(a) Mean availability 70%

(b) Mean availability 50%

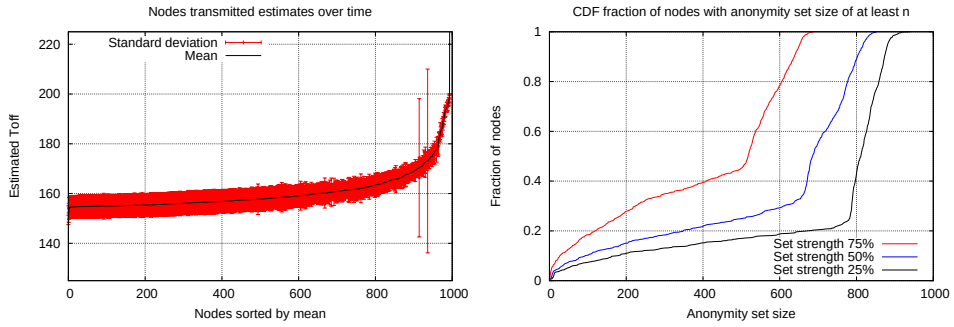
(c) Mean availability 20%

(d) Mean availability 10%

Figure 7.17: Anonymity set size

100 nodes looking to our worst case scenario in Figure 7.17d.

7.5.3 Privacy-preservation under real-life churn using Skype traces



(a) Mean estimate and SD over time

(b) Node anonymity set size CDF

Figure 7.18: Mean estimate and standard deviation and anonymity set size of Skype traces presented in [31]

Calculating the privacy-preserving capabilities of the proposed algorithm under using the real-life churn traces presented in [31] we get Figure 7.18. By our definition the estimates of the Skype trace converged after approximately 20.000 shuffle cycles, so the calculations are means captured from that moment onward. As the estimates them selves fluctuate

over time (see Figure 7.14) the standard deviation of the individual nodes is fairly high compared to the figures we where seeing from the artificial churn (Figure 7.16). The exception are some very low availability nodes for which are online very few times during the experiment and for which we hence have few data points. These nodes have an extreme standard deviation. Regardless, the big standard deviations mean that the anonymity set size of individual nodes gets very high as we can see from Figure 7.18b. As we can see the situation for the 75% set strength the situation is about the same as seen in Figure 7.17d. Only 20% of the nodes have an anonymity set size of less than 100 nodes. The situation is similar for the set strengths of 50% and 25% though more nodes have bigger set sizes than in our artificial scenario.

7.6 Evaluation summary

In this chapter we have shown that our proposed solution for estimating churn in a privacy-preserving manner works in both artificial churn scenarios as well as scenarios created using real-life churn traces. The protocol delivers reasonable accurate estimates within a short time span, and adapts to changing churn values without having to restart or running multiple instances of the protocol. The solution is also privacy-preserving for the vast majority of the nodes in the network, though one might want to take some additional measures for nodes being on the extreme lower edges of the availability scale. The reason for this is mostly that these nodes are unavailable for so long that when they come back online they do not have any valid pseudonyms left in their network overlay. Not transmitting values before one has reached some threshold in number of pseudonyms observation set could be a possible approach. Or only gossiping with trusted nodes until some threshold is reached. But all in all we are feeling confident that our solution is a good one.

Part III

Conclusion and future work

Chapter 8

Conclusion and future work

8.1 Conclusion

In this thesis we set out to take the basis for a protocol which allows for robust privacy-preserving data dissemination over a social graph presented in [33] one step closer to a complete working system. More specifically we set out to show that it is possible to estimate the mean T_{off} time in such a system in an efficient and accurate manner, without compromising the privacy-preserving elements of the system. To the best of our knowledge this is the first protocol which allows for the estimation of churn in large scale P2P network in a privacy preserving manner.

Our solution works by having participating nodes observe their own behaviour patterns in terms of T_{off} time, and then disseminate this information through the network. To do this in a privacy-preserving manner nodes will compute a mean between their own observation about them selves and the observations disseminated to them from their overlay neighbours in the network before gossiping their observation to other nodes. This ensures that the observations disseminated are fairly uniform across all nodes, making it hard or near impossible to track individual nodes based on their observations about themselves. Nodes will finally use the observations received from its overlay neighbours to estimate the mean T_{off} time in the network.

We have shown that the algorithm will converge within a reasonable time given the churn scenario, and once it has converged all nodes will estimate the same mean T_{off} across the system with a very low standard deviation among the estimates. Our solution is also robust as it works in scenarios with high churn as it has the same robustness properties as the work presented in [33] as it uses the same gossiping and sampling techniques.

The accuracy of the estimates will decline somewhat as the churn increases, however this can be remedied by increasing the time in which observations are valid (pseudonym lifetime in this work). Accuracy does however come at the cost of speed of convergence and how fast the system adapts to changes in the observations. Another way of potentially adapting

the system to be more accurate in its estimate would be to vary the sample size. Bigger samples (number of observations) are more accurate both in terms of standard deviation and estimated value, however this also makes the system slower to converge and less responsive to change.

We have also introduced a new metric for evaluating the degree of privacy an anonymity set will provide, based on the probability that two nodes will transmit an indistinguishable numerical value. Having such a generic metric for anonymity sets enables the comparison of the privacy-preserving capabilities of different estimation and aggregation algorithms in a fair way as one can evaluate both the size of the set and the degree of privacy it provides. And as shown in Section 7.5 our protocol preserver node privacy by providing large anonymity sets for the majority of nodes according to our metric.

In this thesis we have concentrated on the estimation of mean T_{off} in the network, however we see no reason why our approach should lend itself to estimating other parameters as well.

8.2 Future work

There are some issues with our protocol which deserve some future attention. There should probably be some upper bound on how long estimates should be valid or a requirement for a certain amount of estimates to be in a node's observation set before it provides an estimate of its own, as suggested in Section 7.2.3.

Equally the privacy-preservation of the low availability nodes with small anonymity sets should be considered further. As it is the protocol provides large anonymity sets for the majority of nodes, but a small minority deserves some future attention to ensure that the estimation protocol does not compromise their privacy preservation as well. Gossiping only over trusted links until a node has reached a certain threshold of pseudonym links could be a path worth exploring.

In this thesis we have concentrated on a solution for estimation of global parameters in privacy-preserving networks. However our approach could be used as an estimation technique for estimating churn also in P2P networks which do not require the privacy preserving capabilities. Especially networks which could utilize the approach proposed by [2] could benefit from our algorithm to get a more uniform estimate across all nodes, but this would have to be explored further. It could potentially also be interesting to utilize our algorithm in systems which uses the algorithm proposed in [15]. The potential benefit would be to have an algorithm that performs better under churn, and without the need for running multiple parallel instances of the algorithm. The drawback is of course that our algorithm requires more complex structures and gossiping methods than the solution proposed in [15].

If limiting ourselves to the work proposed by [33], for which this protocol was designed, using the estimates to preform self-adaptation could potentially be interesting. If a node for instance sees a lot of small

fluctuations in the estimated mean, such as can be noticed in Figure 7.11 when examining the lines for 25 pseudonym links more closely, it could deduce that its number of overlay links is a bit to small for the network and increase the number of pseudonym links it maintains. Similarly if nodes where to observe periodical fluctuations, such as we observe in 7.14, nodes could deduce that there are a lot of nodes which have some very long mean T_{off} times and increase the pseudonym lifetime ratio to achieve better connectivity for these nodes.

Our metric for calculating set strengths could probably also benefit from some future polishing and refinement.

8.3 Acknowledgements

Thanks to Roman Vitenberg, Abhishek Singh and Vinnay Setty for discussions, directions and feedback while working on this thesis. Thanks to my friends and family for being patient with me while writing it, and giving me a push when I needed one. Thanks to Tina for being understanding and supportive. A special thanks to Sabine and Stefan for proofreading and ridding it of most errors. The ones which are left are all my own. Thanks to Realistforeningen and Studievereniging Storm for making student life fun, and helping me find a lot of good friends.

Bibliography

[1] Hari Balakrishnan, M Frans Kaashoek, David Karger, Robert Morris and Ion Stoica. 'Looking up data in P2P systems'. In: *Communications of the ACM* 46.2 (2003), pp. 43–48.

[2] Andreas Binzenhofer and Kenji Leibnitz. 'Estimating Churn in Structured P2P Networks'. In: *Managing Traffic Performance in Converged Networks*. Ed. by Lorne Mason, Tadeusz Drwiega and James Yan. Vol. 4516. Lecture Notes in Computer Science. 2007, pp. 630–641.

[3] Bela Bollobas. *Modern Graph Theory (Graduate Texts in Mathematics)*. Corrected. Oct. 2008.

[4] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot and Alexander Shraer. 'Brahms: Byzantine resilient random membership sampling'. In: *Computer Networks* 53.13 (2009). Gossiping in Distributed Systems, pp. 2340 –2359.

[5] David Chaum. 'The dining cryptographers problem: Unconditional sender and recipient untraceability'. In: *Journal of cryptology* 1.1 (1988), pp. 65–75.

[6] David L Chaum. 'Untraceable electronic mail, return addresses, and digital pseudonyms'. In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[7] Ian Clarke, Oskar Sandberg, Matthew Toseland and Vilhelm Verendel. 'Private communication through a network of trusted connections: The dark freenet'. In: *Network* (2010).

[8] Claudia DÃaz, Stefaan Seys, Joris Claessens and Bart Preneel. 'Towards Measuring Anonymity'. In: *Privacy Enhancing Technologies*. Vol. 2482. Lecture Notes in Computer Science. 2003, pp. 54–68.

[9] Cristiano Giuffrida and Stefano Ortolani. 'A Gossip-based Churn Estimator for Large Dynamic Networks'. In: *Proceedings of ASCI*. Vol. 2010. 2010.

[10] K. Graffi, A. Kovacevic, Song Xiao and R. Steinmetz. 'SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems'. In: *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*. 2008, pp. 279–286.

[11] Vincent Gramoli, Anne-Marie Kermarrec and Erwan Le Merrer. 'Distributed churn measurement in arbitrary networks'. In: *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*. ACM. 2008, pp. 431–431.

[12] Alessio Guerrieri, Iacopo Carreras, Francesco De Pellegrini, Daniele Miorandi and Alberto Montresor. 'Distributed estimation of global parameters in delay-tolerant networks'. In: *Computer Communications* 33.13 (2010), pp. 1472 –1482.

[13] Maya Haridasan and Robbert van Renesse. 'Gossip-based distribution estimation in peer-to-peer networks.' In: *IPTPS*. 2008, p. 13.

[14] Stratos Idreos, Manolis Koubarakis and Christos Tryfonopoulos. 'P2P-DIET: an extensible P2P service that unifies ad-hoc and continuous querying in super-peer networks'. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. SIGMOD '04. Paris, France, 2004, pp. 933–934.

[15] Márk Jelasity, Alberto Montresor and Ozalp Babaoglu. 'Gossip-based aggregation in large dynamic networks'. In: *ACM Trans. Comput. Syst.* 23.3 (Aug. 2005), pp. 219–252.

[16] D. Jurca and R. Stadler. 'H-GAP: estimating histograms of local variables with accuracy objectives for distributed real-time monitoring'. In: *Network and Service Management, IEEE Transactions on* 7.2 (2010), pp. 83–95.

[17] Srinivas Kashyap, Supratim Deb, KVM Naidu, Rajeev Rastogi and Anand Srinivasan. 'Gossip-Based Aggregate Computation with Low Communication Overhead'. In: *Telecommunications Network Strategy and Planning Symposium, 2006. NETWORKS 2006. 12th International*. IEEE. 2006, pp. 1–6.

[18] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell and Seif Haridi. 'A Statistical Theory of Chord Under Churn'. In: *Peer-to-Peer Systems IV*. Ed. by Miguel Castro and Robbert Renesse. Vol. 3640. Lecture Notes in Computer Science. 2005, pp. 93–103.

[19] Jinyang Li, Jeremy Stribling, ThomerM. Gil, Robert Morris and M.Frans Kaashoek. 'Comparing the Performance of Distributed Hash Tables Under Churn'. In: *Peer-to-Peer Systems III*. Ed. by GeoffreyM. Voelker and Scott Shenker. Vol. 3279. Lecture Notes in Computer Science. 2005, pp. 87–99.

[20] Jinyang Li, J. Stribling, R. Morris, M.F. Kaashoek and T.M. Gil. 'A performance vs. cost framework for evaluating DHT design tradeoffs under churn'. In: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 1. 2005, 225–236 vol. 1.

[21] Zhiyu Liu, Ruifeng Yuan, Zhenhua Li, Hongxing Li and Guihai Chen. 'Survive under high churn in structured p2p systems: evaluation and strategy'. In: *Computational Science–ICCS 2006*. 2006, pp. 404–411.

[22] Nick Mathewson, Paul Syverson and Roger Dingledine. 'Tor: the second-generation onion router'. In: *Proc. USENIX Security Symp*. 2004.

[23] Alberto Montresor, Márk Jelasity and Ozalp Babaoglu. 'Decentralized Ranking in Large-Scale Overlay Networks'. In: *Proc. of the 1st IEEE Selfman SASO Workshop*. Isola di San Servolo, Venice, Italy, Nov. 2008, pp. 208–213.

[24] Andreas PFITZMANN and Marit KÖHNTOPP. 'Anonymity, unobservability, and pseudonymity: A proposal for terminology'. In: *Lecture notes in computer science* (2001), pp. 1–9.

[25] Andrei Pruteanu, Venkat Iyer and Stefan Dulman. 'ChurnDetect: A Gossip-Based Churn Estimator for Large-Scale Dynamic Networks'. In: *Euro-Par 2011 Parallel Processing*. Ed. by Emmanuel Jeannot, Raymond Namyst and Jean Roman. Vol. 6853. Lecture Notes in Computer Science. 2011, pp. 289–301.

[26] Yi Qiao and Fabián E Bustamante. 'Elders know best-handling churn in less structured p2p systems'. In: *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*. IEEE. 2005, pp. 77–86.

[27] M.G. Reed, P.F. Syverson and D.M. Goldschlag. 'Anonymous connections and onion routing'. In: *Selected Areas in Communications, IEEE Journal on* 16.4 (1998), pp. 482–494.

[28] Michael K. Reiter and Aviel D. Rubin. 'Crowds: anonymity for Web transactions'. In: *ACM Trans. Inf. Syst. Secur.* 1.1 (1998), pp. 66–92.

[29] Sean Rhea, Dennis Geels, Timothy Roscoe and John Kubiatowicz. 'Handling churn in a DHT'. In: *Proceedings of the USENIX Annual Technical Conference*. Boston, MA, USA. 2004, pp. 127–140.

[30] J. Sacha, J. Napper, C. Stratan and G. Pierre. 'Adam2: Reliable Distribution Estimation in Decentralised Environments'. In: *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. 2010, pp. 697–707.

[31] Neil Daswani Saikat Guha and Ravi Jain. 'An Experimental Study of the Skype Peer-to-Peer VoIP System'. In: *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS)*. Feb. 2006.

[32] Andrei Serjantov and George Danezis. 'Towards an Information Theoretic Metric for Anonymity'. In: *Privacy Enhancing Technologies*. Vol. 2482. Lecture Notes in Computer Science. 2003, pp. 41–53.

[33] A. Singh, G. Urdaneta, M. van Steen and R. Vitenberg. 'Robust Overlays for Privacy-Preserving Data Dissemination over a Social Graph'. In: *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. 2012, pp. 234–244.

[34] Robbert Van Renesse, Kenneth P. Birman and Werner Vogels. 'Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining'. In: *ACM Trans. Comput. Syst.* 21.2 (May 2003), pp. 164–206.

[35] Spyros Voulgaris, Daniela Gavidia and Maarten Steen. 'CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays'. English. In: *Journal of Network and Systems Management* 13.2 (2005), pp. 197–217.

[36] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P.N. Puttaswamy and Ben Y. Zhao. 'User interactions in social networks and their implications'. In: *Proceedings of the 4th ACM European conference on Computer systems*. EuroSys '09. New York, NY, USA, 2009, pp. 205–218.

[37] Praveen Yalagandula and Mike Dahlin. 'A scalable distributed information management system'. In: *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. SIGCOMM '04. Portland, Oregon, USA, 2004, pp. 379–390.

[38] Zhongmei Yao, D. Leonard, Xiaoming Wang and D. Loguinov. 'Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks'. In: *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*. 2006, pp. 32–41.